Robust and stable discrete adjoint solver development for shape optimisation of incompressible flows with industrial applications

Submitted in partial fulfilment of the requirements of the Degree of
Doctor of Philosophy

Yang Wang
School of Engineering and Material Science
Queen Mary University of London

2016

Statement of originality

I, Yang Wang, confirm that the research included within this thesis is my own work or that where it has been carried out in collaboration with, or supported by others, that this is duly acknowledged below and my contribution indicated. Previously published material is also acknowledged below.

I attest that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge break any UK law, infringe any third partys copyright or other Intellectual Property Right, or contain any confidential material.

I accept that the College has the right to use plagiarism detection software to check the electronic version of the thesis.

I confirm that this thesis has not been previously submitted for the award of a degree by this or any other university.

Signature: Yang Wang

Date: 19th September, 2016

Abstract

This thesis investigates stabilisation of the SIMPLE-family discretisations for incompressible flow and their discrete adjoint counterparts. The SIMPLE method is presented from typical "prediction-correction" point of view, but also using a pressure Schur complement approach, which leads to a wider class of schemes. A novel semi-coupled implicit solver with velocity coupling is proposed to improve stability. Skewness correction methods are applied to enhance solver accuracy on non-orthogonal grids. An algebraic multi grid linear solver from the HYPRE library is linked to flow and discrete adjoint solvers to further stabilise the computation and improve the convergence rate. With the improved implementation, both of flow and discrete adjoint solvers can be applied to a wide range of 2D and 3D test cases. Results show that the semi-coupled implicit solver is more robust compared to the standard SIMPLE solver. A shape optimisation of a S-bend air flow duct from a VW Golf vehicle is studied using a CAD-based parametrisation for two Reynolds numbers. The optimised shapes and their flows are analysed to confirm the physical nature of the improvement.

A first application of the new stabilised discrete adjoint method to a reverse osmosis (RO) membrane channel flow is presented. A CFD model of the RO membrane process with a membrane boundary condition is added. Two objective functions, pressure drop and permeate flux, are evaluated for various spacer geometries such as open channel, cavity, submerged and zigzag spacer arrangements. The flow and the surface sensitivity of these two objective functions is computed and analysed for these geometries. An optimisation with a node-base parametrisation approach is carried out for the zigzag configuration channel flow in order to reduce the pressure drop. Results indicate that the pressure loss can be reduced by 24% with a slight reduction in permeate flux by 0.43%.

Acknowledgement

First of all, I would like to express the profound gratitude to my academic supervisor, Dr. Jens-Dominik Müller. This thesis cannot be completed without his endless help. He always be patient and supportive for my research with his wide knowledge and rich research experience. His critical questions and challenges inquire me to think deeply and comprehensively in order to improve my work. Besides, the warm care from him also encourages me and enables me to process my work successfully.

I also wish to express my sincere thanks to my lovely colleagues in School of Engineering and Material Science (SEMS) at QMUL. The valuable discussion with them and suggestions from them always bring about good ideas. The pleasant cooperation and unforgettable working memories is an irreplaceable and wonderful part of my PhD period.

This work is supported by Queen Mary-China Scholarship Council Co-funded Scholarship No. 201206280018. I would like to thank China Scholarship Council and Queen Mary University of London to fund my PhD project.

I would also like to thank my friends. Their care and help lets me feel home-like warmness.

Last but not the least, I would like to thank my family who always support me spiritually throughout writing this thesis and my life in general.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviation

CM Control mass

CV Control volume

AMG Algebraic multi grid

BI-CGSTAB Bi-conjugate gradient stabilized linear solver

BRep Boundary representation

CAD Computer aided design

CDS Central differencing scheme

CFD Computational fluid dynamics

CGSTAB Conjugate gradient stabilized linear solver

CP Concentration polarisation

FD Finite difference method

FVM Finite volume method

GA Genetic algorithm

GMRES The generalized minimum residual method

IDC Improved deferred correction

ILU Incomplete LU decomposition or fractorization

LCO Limit-cycle oscillation

NF Nano-filtration

NsPCC NURBS-based parametrisation with continuity constraints

NURBS Non-uniform rational B-splines

PSC Pressure Schur complement

RO Reverse osmosis

SAI Sparse approximate inverse method

SOR Successive over-relaxation linear solver

SWM Spiral wound membrane

TVD Total variation diminishing schemes

UDS Upwind differencing scheme

Sub-/ Superscript

I The centroid index of control volume I

J The centroid index of control volume J

k The index of control volume face

Variables

$\dot{f}$ Mass flux

μ The fluid dynamic viscosity

$\mathbf{n}$ The unit normal vector to control volume face

Ω The size of finite/control volume

ϕ Arbitrary scalar or velocity component

ρ The fluid density

$\mathbf{u}$ Fluid velocity

$\mathbf{u}_{CV}$ Control volume velocity

A_I The coefficient of variables at centroids of control volume I

A_J The coefficient of variables at centroids of control volume J

p Fluid static pressure

S The face area of control volume

CBL Convection blending coefficient

Chapter 1

Introduction

With continued improvement of computer technology, the increase in calculation speed and memory, Computational Fluid Dynamics (CFD) has become an important tool in academic research and industrial applications in recent decades. Turbomachinery internal flow simulation and external aerodynamics of vehicle computation are common CFD applications in industry. Recently scientists, like biomechanist and biomedical scientists, study body fluids with CFD as auxiliary approach [1, 2]. In chemical engineering, CFD has become a popular method to investigate and analyze fluid process and accordingly new CFD models are also proposed for solving micro scale flow cases. CFD studies have expanded to a big range from complex large scale flows structure to micro scale detailed flow mechanisms. Numerous software sources development and hardware enhancement has strengthened the capacity of CFD to solve more and more complex flow problems. With the maturing of CFD, numerical solutions become reliable and can be more efficiently obtained in a cheaper way compared with expensive experiments, and CFD offers valuable data as reasonable prediction and evaluation for laboratory equipment set-up. Besides, CFD can conduct theoretical approach to obtain more accurate results along with experimental data to analyze physical foundation. Therefore, this numerical methodology plays a significant role in exploring flow issues as we can see from the diversity of commercial CFD software, such as Fluent [3], CFX [4], STAR-CD [5] and others to numerous open-source CFD tools, like OpenFoam [6], SU2 [7], Dolfyn[8] and so on. Each of them has a certain volume of users, which indicates that CFD is being applied widely for many flow relevant studies. Based on the thriving of numerical methods and computational high fidelity with increasing cost reduction, some researchers have shifted their attention from flow evaluation to problems of optimization. And numerical optimization methods with CFD can be integrated seamlessly to build up a complete process in order to pursue optimal solutions. In practical optimization cases, shape optimization is considered to be a substantial part in industrial design, and related toolboxes for shape optimization are developed and introduced progressively

into lots of CFD codes. Among them, methods for calculating surface sensitivity are widely developed for gradient-based shape optimization. The sensitivity expresses the relationship of shape changing with respect to design parameters, normally the shape geometry data.

This project focuses on developing a robust and stable in-house CFD-based optimisation code with reliable incompressible flow computation and sensitivity information using discrete adjoint method.

1.1 Background of optimisation in CFD

In 1933 the British scientist Thom calculated the flow past a circular cylinder via by hand-operated computer [9], from which, this numerical field has advanced for several decades in parallel with advances in computer science and hardware technologies. The state of the art in CFD includes higher-order schemes, automatic mesh generation and highly efficient algebraic solvers. CFD has come to a high speed development period in history.

Rapidly improvement of CFD spurs us to pursue the optimal solution all the time. Numerical optimisation method recently have drawn a great deal of attention in the industrial design area. And design for fluid flow plays an significant role in a wide range of engineering issues including aeronautic [10, 11], turbo-machinery [12, 13] and automotive design [14, 15]. In contrast to traditional optimisation methods relying heavily on the experience and intuition of a human designer, computation in pursuit of the optimum via numerical algorithms has its evident vantages.

There are various numerical algorithms in practical engineering optimisation. From the view of optimisation path, they can be classified as 'stochastic' and 'deterministic' methods. The major difference between the stochastic algorithms and deterministic algorithms is the methodology of searching the updating direction at every-iteration. Typical stochastic algorithms include genetic algorithms (GA), evolutionary algorithms; and in the category of deterministic algorithm, there are response-surface methods, meta-modelling, simplex methods, gradient-based, etc. [1]. Gradient-based optimisation algorithms search the optimum in the direction of gradient or based on gradient information at every-iteration. Because gradient presents the maximum changing rate of objective function, gradient-based methods usually significantly increase the speed of convergence compared to other methods which do not use gradient. Therefore, rapid convergence is a primary advantage of the method. With the more accuracy of gradient calculation, more fast convergence can be achieved normally. However, the gradient-based algorithms tend to find a local

[1] `http://www.sems.qmul.ac.uk/courses/lecture.php?id=5601`

minimum depending on the topology of the objective function and the initial state of the optimisation. In contrast, when searching for a global optimum of a discontinuous or noisy function, methods like genetic or evolutionary algorithms are usually adopted. Stochastic algorithms search toward the global optimum by evaluating a population-based of solution candidates. Using stochastic algorithm, the updating direction or the optimum direction of the current iteration is usually determined stochastically based on the solutions of the latest or several latest iterations. Genetic algorithms are inspired by the natural evolution. GA treats the function evaluation as a black box and the searching direction is found using a number of operators in a population of solution candidates, mainly including inheritance, crossover, mutation and selection.

Zingg and et al. compared GA and a gradient-based (adjoint) algorithm in several cases of aerodynamic shape optimisation problems, including a single-point shape optimisation, a multi-point shape optimisation and a multi-objective optimisation [16]. Through the systematic comparisons, besides advantages and disadvantages of these two algorithms, they also pointed out [16]: GA (as a stochastic method) is suitable for preliminary design with low-fidelity models to explore a wide range of design space, while the gradient-based algorithm can be more appropriately used for detailed design with high-fidelity models to modify the existing designs under a narrower consideration. The focus of this thesis falls into the latter, which is the detailed design for further improvement of the currently used shape. In fact, after several hundreds of years of Industrial Revolution, significant improvements have been achieved in a number of mature technologies from generation to generation. In turbo-machinery, a well-designed machine model has achieved sufficiently high efficiency through the many years' development. If we want to further enhance the performance by improving the internal flow, the shape optimisation of blade would not be far away from the baseline design. Similarly, in other applications of industrial design, the modification of the current module/model would be a big volume. Therefore, the gradient-based approach is preferred for dealing with these small deformation problems on the efficiency and accuracy.

In this context, the adjoint approach is the most efficient method to calculate the gradient as the calculation is almost independent of the number of design variables. The method originates from control theory, which solves an additional adjoint equation using coefficients determined by the solution of the primal equations. The method has been successfully used in aerodynamic designs [17–19] and other applications of sensitivity analysis [20]. While the adjoint approach is widely used in sensitivity analysis and gradient-based optimisation in compressible flow based industrial aerodynamic applications, this optimisation approach is much less developed for incompressible flow based cases.

Adjoint optimisation considered in this thesis is a Navier-Stokes (N-S) Equation constrained optimisation problem. The convergence of the flow solver has a significant impact on the performance of the adjoint solver as it inherits linear stability/instability from the non-linear flow solution. The adjoint solver can diverge if the iterative algorithm of the non-linear flow solver converges only to limit cycles [21, 22]. The instability of the flow solver can be caused by both physical and numerical factors. Unstable flow, such as flow separation, produces the inherent instability in the linearized equations and results in the difficulty of solver to be converged asymptotically. Besides the inherent physical instability, there are also other factors affecting the quality of the linearized system of the non-linear flow when the system is generated and solved. To this end, the discretization scheme, mesh quality, pre-processing, and linear solver are all influencing factors to determine the convergence of the non-linear flow. In incompressible numerical methods, the classical staggered grid discretisation is more difficult to implement accurately on general unstructured grids than collocated grid methods [23]. Non-staggered schemes call for appropriate interpolation of variables such as pressure. Also, mesh quality and quantity significantly influence the generation of a linearised system. On one hand, a highly refined mesh which resolves the unsteadiness of the local flow increases the difficulty to reach asymptotic convergence using a steady flow solver. On the other hand, the conservation of the equation deteriorates if the mesh is considerably skewed because inaccurate fluxes accumulate due to the mesh skewness. Furthermore, when a linearised system is formulated from a non-linear flow, how to solve the system is also very important. A number of pre-conditioning methods and linear solvers are available to deal with different types of systems. Therefore, stabilisation strategies of an unstable non-linear flow are essential to ensure the availability and accuracy of the optimisation loop based on the adjoint method.

1.2 Development of incompressible flow solver

The Navier-Stokes equations describe the real physical phenomena and solutions of these equations help to understand nature. To solve these equations, a high performance of the CFD software is needed with satisfactory accuracy and efficiency. Particularly, incompressible flow problems attracted attentions for research. For mathematicians, incompressible flow is a perfect playground to test the numeric of dealing with the non-linearity and the velocity divergence-free constraint that leads to the saddle-point problem. For engineers, scientists and software developers, the incompressible flow solver is applicable to many flow problems from liquid flow to low Ma number gas flow. To develop the solver, three reasons that affect efficiency and accuracy of a CFD code were analysed by Turek [24]: bad efficiency of the discretiza-

tion and solver, unnecessary large number of grids and time steps, and inefficient code implementations.

The first reason is associated with discretisation method and numerical algorithm to deal with the non-linear equations. In CFD simulation, generally, there are two types of flow solvers, density-based and pressure-based solvers. Density-based flow solver is utilised widely for computation of steady and unsteady flows whose Mach number is larger than unity. In the subsonic flow problems, when Mach number is low, the dominance of convection term within the equation system renders the system stiff, which makes the solver converge slowly [25] and may suffer severe stability and accuracy restrictions and become inefficient for the low Mach number flows [26]. In this context, pressure-based methods are conceived to solve incompressible flow, which is also applicable for low Mach number flows [27]. These methods use pressure as a primary variable and update both velocity and pressure using pressure correction at every-iteration. Because the pressure variations are always finite irrespective of the flow Mach number, pressure-based approaches show advantages over the density-based methods in terms of the range of flow Mach numbers.

Pressure-based flow solver was first proposed by Chorin [28, 29] using finite difference method (FDM) and then widely and successfully used within finite volume implementations. The finite volume method (FVM) is the most widely utilised numerical techniques in CFD, which refers to a small volume surrounding each node on a mesh [30]. In FVM, volume integrals in N-S equations are converted to surface integrals using the divergence theorem. Thus, these derived terms are evaluated using surface fluxes through the whole mesh. The flux of a surface contributes equally in an identical value entering one adjacent volume and leaving the other one, and ensures the conservation of the method. The most famous class of pressure-based algorithms implemented in FVM is SIMPLE (Semi-Implicit Method for Pressure Linked Equations) family [31]. Many commercial codes depend on SIMPLE based pseudo-time stepping such as FLUENT.

All algorithms in the SIMPLE family, including SIMPLER [32], SIMPLEC [33] and etc were initially developed to solve the velocity-pressure decoupling problem with staggered grids, where two series of mesh nodes are required for velocity and pressure respectively. However, staggered grids require interpolations and an expanded memory system [34]. Faced with complex geometries, collocated grid was proposed [35] and Peric et al. pointed out some obvious advantages of collocated grids over staggered grids [34]: 1) one set of control volumes having the same location to store all variables; 2) coefficients contributed by the convection are same for all variables; 3) Cartesian velocity components can be used in conjunction with non-orthogonal coordinates to deal with complex geometry; 4) fewer constraints on the grid, for example, there is no need to evaluate the so-called "curvature term".

Through the comparisons, the collocated grid shows no disadvantages relative to the staggered grid and converges faster on several test cases [34].

Secondly, size of the mesh and mesh quality have significant effects on the performance of a flow solver. The number of mesh cells influences the accuracy of CFD results. The mesh should catch the flow of interest in a discrete manner with satisfactory details to ensure CFD modelling at the desired accuracy level to be carried out. And the mesh quality influences the convergence during the iterations. It may include cell skewness, cell size variation and etc., which lead to convergence and stability issue. In addition, as it is difficult to determine how good (or bad) a mesh is before carrying out a CFD simulation, a stable and robust solver needs to be capable of converging on a mesh with poor quality. Third, how to implement the details of the discretisation, spatial and time-stepping, obviously affects the performance of the flow solver. As performance of a CFD simulation largely depends on the availability and management of computational resources, a well-developed CFD code is also essential.

In addition to the instability inherited in the governing equations' physical attributes, numerical algorithm and mesh quality, how to solve the derived equations also involves the instability issues. Numerical methods including pre-conditioning and linear iterative algorithms are capable of changing the quality of the derived linear system and its convergence. A preconditioner is typically used to reduce a condition number of the problem by transferring the system one into another system which has better properties for iterative linear solver. Especially for matrices of large dimension, it can be very beneficial to have a good matrix approximation and improve the convergence within the available computational resources. Preconditioning is widely used with Krylov subspace iterative approaches [36–38]. For a problem with a symmetric positive definite matrix, the eigenvalues of the matrix determine the rate of convergence of the conjugate gradient (CG) method. Preconditioned CG (PCG) aims to manipulate the all the eigenvalues clustered around 1. In contrast, convergence of a non-symmetric problem is more complex since the eigenvalues may be not the determinant to the non-symmetric matrix iterations (e.g. GMRES). Nevertheless, preconditioning still plays an important role to accelerate the convergence especially when the preconditioned matrix is close to normal. Benzi presented a survey of preconditioning techniques for improving the performance and reliability of Krylov subspace methods [39]. The main algebraic discussed are incomplete factorization techniques, sparse approximation inverses, and algebraic multilevel methods [39].

1.3 Adjoint sensitivity solver for optimisation

The utilisation of adjoint equation originates from the optimal control theory [18, 40, 41]. Combined with Stokes equation, the first appearance of adjoint methods was presented by Pironneau in 1970's [42]. In 1980's and 1990's, the popularity of adjoint method started within the research of CFD due to a number of Jameson et al.'s pioneered works on adjoint method in aerodynamic optimal design [17–19, 41, 43–49], in which they applied the adjoint method to flow equations from 2D Euler equations to 3D N-S equations.

Applications of the adjoint method in fluid dynamics are mainly in the field of aerodynamics. After the introduction of the adjoint method to aerodynamics by Jameson in transonic flow, and inviscid compressible flows with shock waves [43, 50, 51], Jameson et al. used the adjoint method to design an aircraft in both viscous transonic [18] and supersonic flows [47].

Compared to the application of the adjoint method in aerodynamics, adjoint applications in incompressible flow problems, such as automotive industries, lag behind [52]. Othmer and Villiers presented a topology optimisation of 3D air duct manifold using the continuous adjoint approach [53]. Xu et al presented work using discrete adjoint flow solver to compute the sensitivity of the CAD variables and demonstrated the method by carrying out a 3D segment of a Volkswagen air duct [22].

Besides these conventional applications of the adjoint solver, it has been also successfully utilised in other applications. Previous literatures in flow equations constrained sensitivity analysis and optimisation problems in heat transfer and array arrangement problems are particularly interested in this work. Marck et al. carried out an optimisation of both fluid dynamics and heat transfer in the frame of laminar flows using FVM and discrete adjoint method [54]. With gradient computed using the adjoint method, Oevelen and Baelmans maximised the cooling of an isothermal heat source by optimising the location of fins and channels [55]. Kontoleontos et al. extended the porosity-based adjoint method to account for heat transfer problems in turbulent flow [56]. Yan and Gao developed a study of the shape optimisation problem in Navier-Stokes flow coupled with convection heat transfer using a gradient-based algorithm based on adjoint solver [57]. Alexander et al. optimised design of heat sinks and micro-pumps based on natural convection effects using discrete adjoint sensitivity analysis [58]. Furthermore, the adjoint approach can be found to be successfully for optimisation of arrangement of array of physical obstructions in flow. Funke et al. developed an optimisation of arranging tidal turbine array to maximum power extraction using the adjoint approach [59, 60].

There are two approaches to develop the adjoint equations, including the continuous adjoint and discrete adjoint. In both approaches, the starting point is the

analytical form of primal equations which is incompressible N-S equations in this thesis. As illustrated in Fig. 1.1, the differences between the two approaches are how the adjoint equations are derived and how the codes are implemented. In the continuous adjoint approach, the primal equations combined with cost function are first linearised, where the adjoint equation is derived at the same time. Then, the adjoint equations are discretized. Conversely, in the discrete adjoint approach, the primal equations are first discretized and then linearised. The resulted primal equations are finally transposed to generate the discrete adjoint code.

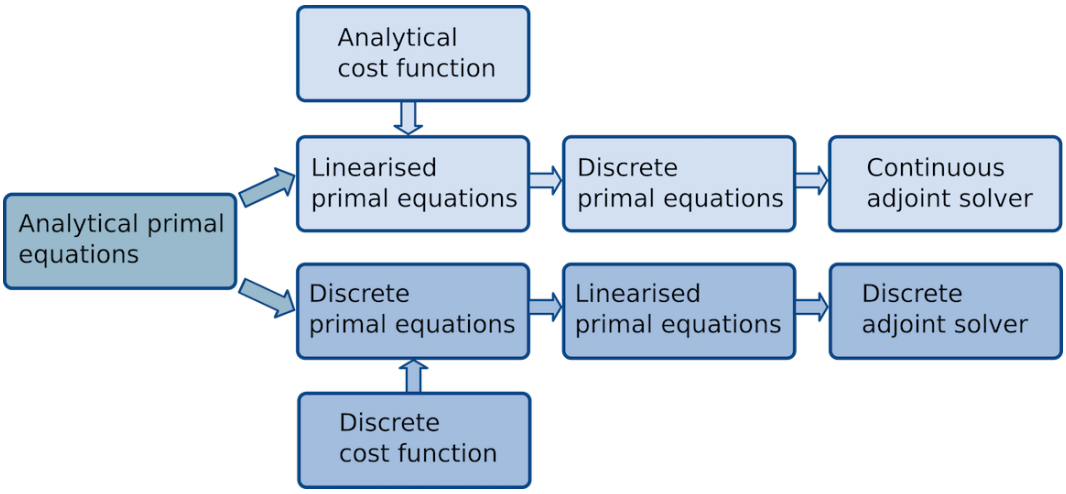


Figure 1.1: Continuous adjoint approach vs. discrete adjoint approach

With the limit of infinite grid resolution, theoretically, both the two adjoint approaches converge to the same result. But due to the different derivation and implementation, there are distinctive features between them. There can be found several discussions of the continuous adjoint and discrete adjoint in the literature [61–63]. Generally, the advantages of the continuous one are low memory consumption and straightforward implementation of the adjoint equation with the primal equations. A continuous adjoint solver has been successfully added in OpenFOAM [53]. Because the cost function is involved in deriving the boundary conditions, the continuous adjoint approach needs to be analytically considered in the derivation for every optimisation case with different objective function. In contrast, major advantages of the discrete adjoint approach are the exact gradient of the discrete cost function and the fully converged optimisation process. This allows a check on the correctness of the implementation by comparing with the other acceptable methods, such as finite difference and tangent linear solutions. In addition, the discrete adjoint code can be developed using automatic differentiation (AD) tools which lead to automation of the derivation in the adjoint equations.

AD is a collection of methods to evaluate derivatives of differentiable functions

through representations of programming[64]. The basics of AD tools are all the programs can be divided into a sequence of elementary operations (addition, subtraction, multiplication, division and etc.) and elementary math functions (exponential, logistic, sine, cosine and etc.). Accordingly, the derivatives can be calculated by repeatedly applying the chain rule as accurate as the computers precision. There are two main ways to design and implement AD tools.

Source code transformation: the derivatives are calculated by explicitly building a new source code. This method parses the original codes, build an internal representation, and generate the differentiated codes [65]. This allows the AD tools to analyse the original code globally and optimise the generated differentiated code.

Operator overloading: with the allowance of the language, the derivatives are calculated by overloading the arithmetic operations by replacing the types of the floating point variables with a one that has additional derivatives [66]. The overloaded types and arithmetic operations need to follow the structure of the original code. Within the same implementation framework, the overloaded library can be easily redefined for different purposes. Hascoet and Pascual pointed out [67]: operator overloading is fine for tangent mode but implies some acrobacy for the adjoint mode. On the other hand, code transformation is a better choice for adjoint mode of AD in which control-flow and data-flow reversal and global data-flow are important.

In this context, using the discrete adjoint approach, automatic creation of adjoint programs is achieved by using AD tools. Because of the transposed matrix has the same eigenvalues of the linearised primal system, the same convergence of the iterative method is guaranteed in the discrete adjoint system. Therefore, the quality of the linearised matrix of the non-linear primal equations and the performance of the flow solver are essential to determine the stability and convergence of the discrete adjoint linear solver. If the convergence of the primal equations is fast, the same performance of the discrete adjoint solver can be expected by using the same iterative methods.

1.4 Shape optimisation of spacer in reverse osmosis membrane process

As the adjoint method has started to be considered as a standard tool in design and optimisation of PDE-based flow problems, especially for N-S equations based compressible flow problems in aerodynamics, this study aims to explore a new application of adjoint method in shape optimisation of spacer in membrane process and take reverse osmosis (RO) as a case study to demonstrate the applicability of the

method.

Generally, membrane-based filtration process with spiral wound membrane (SWM) module is now a standard unit operation [68]. The major problems that reduce the performance of various membrane filtration processes are mainly concentration polarisation (CP) [69] and membrane fouling and scaling [70]. Due to the salt rejection of the membrane, CP occurs in the boundary layer because of the rapid convection of solute to the membrane surface by permeation and slow back diffusion. Membrane scaling or fouling is the permanent deposition of species on the membrane surface or in the membrane support layer, which reduce the membrane permeability and permeation flux. It is important to investigate membrane filtration processes in order to not only increase mass transfer but also control membrane fouling, considering system energy consumption.

Spacers in membrane channels are basic functional components of the SWM modules, significantly influencing the flow pattern and membrane module performance. In general, spacers are obstacles to flow in the channel, which not only promote eddy and vortex but also result in increased pressure loss through the channel. They generate strong shear stress along the boundary layer at the membrane surface and enhance back-mixing of concentration polarisation by high flow velocities or secondary flow patterns. Characterisation of a good spacer may include 1) efficiency in promoting vortices in flow; 2) efficiency in controlling scaling and fouling; 3) capability to provide mechanically support; and 4) controlling pressure drops to an acceptable level. However, the optimum design of spacers depends on many factors such as feed velocity, flow properties, and channel dimensions. Therefore, a large number of both experimental and numerical investigations associated with design, selection and optimisation of spacers in the membrane channels have been carried out, which mainly include spacer arrangement, spacing and shape. Geraldes et al. carried out a series of experimental and numerical works of CP in SWM with ladder-type spacers in which two-layers spacers were utilised [71]. The longitudinal filaments define straight channels to reduce the pressure drop and the transverse filaments disrupt the boundary layer and minimise CP [71–73]. With experimental friction factor, by CFD simulation, Santos et al. investigated twelve flow-aligned spacer structures which were made by varying the distance between the centre of consecutive transverse filaments and the number of filament located along the longitudinal direction in the range of Re between 50-600 [74]. Results showed insignificance of the number of longitudinal filaments affecting the flow profile, indicating only the transverse filaments determine the pattern of flow and mass transfer in the spacer-filled channel [74].

In general, three main arrangements of transverse filaments, namely zigzag, cavity and submerged (shown in Chapter 6), have been widely investigated. Contact of each zigzag or cavity filaments with a membrane or wall promotes the formation of flow

mixing zones both upstream and downstream of the filament-membrane/wall contact lines. Conversely, because submerged filaments are symmetrically located in mid-plane, these filaments reduce the cross-section area of flow and cause the evolution of closed vortices behind each filament. Although this flow pattern is steady at low Re number and relatively far away from the membrane surfaces, the vertices are easy to oscillate with respect to the increase of Re number and interact with boundary layers near the membrane. Schwinge et al. compared zigzag, submerged and cavity transverse filaments arrangement in terms of mass transfer and pressure loss [68]. It was found that for mass transfer zigzag arrangement achieved the highest flux and cavity arrangement is better than submerge spacers. In contrast, for pressure loss, the submerged spacer showed the highest pressure loss [68]. Wardeh and Morvan also pointed out that zigzag is more efficient and economical in removing accumulated salt from the membrane surface compared to submerged [75]. Ahmad et al. stated more turbulence effect is generated by presenting the filaments adjacent to the membrane as compared to the case where by the filaments are located opposite to the membrane [76]. Shakaib et al. studied a novel spacers with filaments inclined towards channel axis (which is similar to zigzag) and also found enhanced mass transfer rate[77]. But the contact lines of spacers-membrane interface in zigzag and cavity are highly likely to lead to membrane fouling due to the "shadow effect" that can accumulate foulant on the membrane surface or spacer [78]. Additionally, spacer spacing were also studied. According to Saeed et al.'s work, it indicated that mass transfer coefficient values were not significantly different for the spacers having low to moderate filament spacing but sharp decline in both pressure drop and mass transfer coefficient were obtained when the filament spacing increased to a particular level [79].

Apart from spacer arrangement, spacer shape or geometry is also a hot topic in the field. Ahmad et al. pointed out that triangular filaments demonstrated the highest degree of CP reduction ability and pressure drop followed by square and circular filaments [76]. Thus, it was recommended to utilise triangular filaments at low Re numbers and circular at high Re numbers. But Guillen and Hoek carried out a study of feed spacer geometry on NF (nano-filtration) and RO processes and they found that feed spacer geometry had little impact on mass transfer over the selected geometries, including cylinder, ellipse and wing-shaped spacers [80]. Amokrane et al. studied the impact of new spacer designs elliptic and oval shapes on CP effect, mass transfer and pressure drop, and results shows that elliptic and oval shapes generates lower pressure drop compared to conventional spacer geometries and tilted oval spacer changes the flow structure in the vicinity of spacers which is potentially beneficial to prevention of fouling [81]. Furthermore, Dendukuri et al. showed that spacers with concave surfaces give significant reduction in pressure drop across the membrane channel as compared to commercial spacers which have the convex surfaces [82]. Similar concave

spacer shapes are demonstrated to potentially give higher wall shear rates per unit mass of energy dissipation rate [83]. In addition, Liu et al. experimentally studied a novel saw-tooth promoter in flat-sheet membrane module and showed the improved mixing of fluid and back-transport of particles. The shear rates in module with saw-tooth spacers were above 1.98 times higher than that in spacer-free module and the saw-tooth spacer can increase the flux by 115.7%-258.0% with a reduction of specific energy consumption above 33.8% [84]. They further compared the saw-tooth spacers with conventional zigzag configuration with cylinder filaments and found that averaged membrane shear rates in the saw-tooth cavity configuration were more than 6.38 times higher than that in cylinder zigzag configuration at the feed Re number ranging from 750 to 1875 [85].

However, from previous studies focusing on spacer design and selection, almost all the work evaluated the performance of a particular design of spacers or compared several spacer designs by carrying out numbers of experiments or/and numerical tests. There is no methodology of how to optimise the spacers with respect to the objective function. Consequently, existing optimisation of spacer shape and arrangement are achieved by comparing to a finite number of pre-defined case studies, which is time-consuming and experience-dependent. Furthermore, taking advantages of 3D printing, current concerns of feasible conventional manufacture techniques such as vacuum foaming or extrusion are potentially eliminated, which calls for novel designs of spacers with complex shape and non-recurring patterns [78]. It is very difficult to use the design loop of "try and evaluate/compare" to explore the previously infeasible designs of complex and non-periodic structure of spacers. To this end, a new design loop of objective oriented methodology is needed to be developed for design of spacers.

1.5 Motivation and objective

As introduced earlier, the stability of the discrete adjoint solver significantly depends on that of the primal flow solver. Compared to continuously development and improvement on the compressible flow based discrete adjoint solver, investigations on the incompressible flow based discrete adjoint solver have just started since recent years. In addition, to enhance the efficiency of the gradient-based optimisation loop, a high performance of flow solver is essential. The gradient-based optimisation loop asks for the primal solver to be fully converged to machine precision with a relatively low computational cost, which usually cannot be achieved by current standard incompressible flow solver. To improve the stability and robustness of the discrete adjoint solver, therefore, the motivation of this research project is to improve the convergence of the incompressible flow solver. Furthermore, a novel application of

adjoint-based optimisation in an incompressible flow application is carried out. Using the developed discrete adjoint solver, sensitivity analysis and optimisation of spacer shape in the RO membrane channels are developed.

The objectives of this work are therefore,

1) In the framework of stabilising the incompressible flow solver, methods and strategies are developed to treat the velocity-pressure coupling system through the control of the outer and inner iterations.

2) Besides, schemes are implemented in order to strengthen the solver's capacity for skewed mesh. A more robust and stable flow solver is developed to work on a wide range of cases and leads to reliable sensitivity computation via discrete adjoint solver.

3) Discrete adjoint with the improved implementation in tandem with flow solver is also developed with more efficient performance costing less CPU time.

4) Gradient-based shape optimisation is conducted to demonstrate the adjoint involved design loop to achieve improved objective under the perturbation of design variables. Corresponding flow fields analysis at different Reynolds number is given to illustrate the physical explanation for optimised shape.

5) Discrete adjoint-based method is applied to an unexplored industrial realm. By introducing sensitivity analysis to a segment of SWM module, adjoint-based optimisation becomes a novel tool for spacers design. In-house flow and discrete adjoint solvers are developed and validated for membrane process RO flow simulation. Sensitivity of three common spacers' configuration, submerged, cavity and zigzag is computed and physical analysis is elucidated combing with flow fields solution. Permeate flux and pressure drop and respective sensitivity at different Reynolds numbers are compared to show the influence of flow pattern. Optimisation of spacers' shape are carried out based on the gradient information.

1.6 Thesis organisation

Chapter 2 presents the mathematical model of incompressible flow and numerical methods, containing discrete schemes and linearisation algorithms implemented in flow solver GPDE. Components constructing flow solver are also introduced. Two dimensional lid-driven cavity benchmark cases are carried out for the codes validation; For three dimensional validation, solutions are compared with widely used open source CFD code OpenFOAM.

In Chapter 3, discrete adjoint method is expounded. The way of how to manipulate Automatic Differentiation tool Tapenade to build discrete adjoint solver is indicated. Modification of AD solver implementation is introduced for segregated pressure-based flow solver. Sensitivity calculation using the discrete adjoint solver is verified with the finite difference and tangent linear results.

Based on Chapter 2 and Chapter 3, stabilisation strategy are investigated in Chapter 4. SIMPLE-family algorithms are studied from two points of view and four pressure Schur complements including SIMPLE-family algorithms are compared to show the stabilization effects. Advanced liner solver AMG from HYPRE library linked to SIMPLE shows linear system effects for non-linear stabilization. Besides, Semi-coupled Implicit solver with multi stabilization strategies to couple velocity components is proposed.

In Chapter 5, a complete optimisation frame work is constructed for a three dimension industrial s-bend duct case, where CAD-based optimisation is applied. Different flow shows respective shape changing and the optimum shapes are discussed to show the physical flow effects.

Membrane process of RO is studied in Chapter 6. CFD models of RO membrane process are built up first. Flow solver is verified with previous numerical solutions and experiment data in literatures. Discrete adjoint solver is validated for accurate sensitivity solution. A thorough sensitivity analysis of the spacer shapes is carried out to locate the sensitive position to influence the minimization of pressure drop and the maximization of permeation. Finally, an optimisation case with respect to pressure loss of the spacer shape is developed and analysed.

In Chapter 7, the main findings of this work are summarises. Based on results obtained in previous chapters and experience during the study, conclusions and limitations are discussed. Prospective research is proposed as well.

Chapter 2

Incompressible flow solver: GPDE

Incompressible flow is a flow in which the density of a flow does not change in time as it moves through space. It is a property of the whole flow rather than a particular fluid element. Although there is no absolute incompressible flow in practice, compressible flow can be approximated with satisfied accuracy when Mach number is less than 0.3. Mach number, $Ma = \frac{U}{c}$, corresponds to the ratio of fluid velocity U to the speed of sound c. Therefore, applications with low speed flow in which Mach number is far below sonic speed are treated as incompressible flow. In nature and human life, low speed phenomena widely exist, such as ground vehicles and marine vehicle. Ground vehicles are normally driven in the range about 10-300 km/h; while speed of sound is approximately 343.3 m/s (in dry air of a temperature of 20°C)[86], or 1235 km/h. Because of high sound speed in liquid (1481 m/s in water of a temperature of 20°C)[87], marine vehicle is also considered driven at low Mach number. Therefore, lots of industrial flow problems and academic research, studying flow $Mach < 0.3$, can take incompressible flow assumption. In addition, series of mathematical issues come up with incompressible flow at the same time.

In-house CFD code GPDE is an unstructured steady Navier-Stokes solver for incompressible viscous flow. It is developed for exploring N-S equations constrained incompressible flow problem to fulfil the gradient-based optmisation loop. Therefore, there are two main components in GPDE calculation loop: incompressible flow solver and sensitivity solver. To solve the steady incompressible flow equations, on one hand, finite volume method (FVM) is applied for spatial discretisation and SIMPLE[88] algorithm is used to solve the discretised systems by decoupling the velocity and pressure of fluid in a segregated way. On the other hand, sensitivity of the control variables subject to the objective function is calculated using discrete adjoint method.

2.1 The governing equations

Rather than tracking individual fluid particle movement, investigation of flow properties often focuses on flow fields distribution across a certain objective domain, where fluid can flow in and out. For a given open system, the balance of its mass and momentum can be obtained based on the conservation principle that those are neither created nor destroyed. Giving an intensive property[1] ϕ (for mass conservation, $\phi = 1$; for momentum balance, $\phi = \mathbf{u}$; for conservation of a scalar, ϕ presents the conserved property per unit mass), the *control volume equation* or the *Reynolds' transport Theorem* can be expressed as:

$$\frac{d}{dt}\int\limits_{\Omega_{CM}} \rho\phi d\Omega = \frac{d}{dt}\int\limits_{\Omega_{CV}} \rho\phi d\Omega + \int\limits_{S_{CV}} \rho\phi(\mathbf{u} - \mathbf{u}_{CV}) \cdot \mathbf{n} dS \tag{2.1}$$

where ρ is the density of fluid; ϕ could be any conserved property; Ω_{CM} is volume occupied by control mass (from Lagrangian view); Ω_{CV} and S_{CV} are the control volume and control volume surface (from Euler view); $\mathbf{u}$ and $\mathbf{u}_{CV}$ stand respectively for fluid velocity and control volume velocity; $\mathbf{n}$ is the unit normal vector to face S_{CV}.

For a fixed CV, as we consider here, $\mathbf{u}_{CV}$ equals zero. Replacing ϕ by $\mathbf{u}$ and considering external forces, including surface and body forces, we transform the CV equation Eq. (2.1) to the momentum balance equation:

$$\begin{aligned}\frac{d}{dt}\int\limits_{\Omega_{CM}} \rho\mathbf{u} d\Omega &= \frac{d}{dt}\int\limits_{\Omega_{CV}} \rho\mathbf{u} d\Omega + \int\limits_{S_{CV}} \rho\mathbf{u}\mathbf{u} \cdot \mathbf{n} dS \\ &= \Sigma\mathbf{f}\end{aligned} \tag{2.2}$$

where ρ is the density of fluid; Ω_{CV} and S_{CV} are the control volume and control volume surface; $\mathbf{u}$ stands for fluid velocity; $\Sigma\mathbf{f}$ includes body forces (e.g. gravity) and surface forces (e.g. pressure).

Similarly, replacing ϕ by 1 in Eq. (2.1) for fixed CV, and mass balance equation for this open system is:

$$\frac{d}{dt}\int\limits_{\Omega_{CM}} \rho d\Omega = \frac{d}{dt}\int\limits_{\Omega_{CV}} \rho d\Omega + \int\limits_{S_{CV}} \rho\mathbf{u} \cdot \mathbf{n} dS \tag{2.3}$$

[1] An intensive property is a bulk property, meaning that it is a physical property of a system that does not depend on the system size or the amount of material in the system.[89]

For steady case, $\frac{d}{dt} = 0$, momentum and mass conservation is further derived as:

$$\int_{S_{CV}} \rho \mathbf{u}\mathbf{u} \cdot \mathbf{n} dS = \Sigma \mathbf{f} \tag{2.4}$$

$$\int_{S_{CV}} \rho \mathbf{u} \cdot \mathbf{n} dS = 0 \tag{2.5}$$

To derive the system of govern equations of incompressible flow, several assumptions are applied:

- incompressible viscous flow based on continuum medium model:

$$\rho = \text{const}; \tag{2.6}$$

- only viscous stress force and pressure are considered as the surface forces and on body force acts on a control volume:

$$\Sigma \mathbf{f} = \int_{S_{CV}} \tau \cdot \mathbf{n} dS - \int_{S_{CV}} \nabla p \cdot \mathbf{n} dS \tag{2.7}$$

- applied for isotropic Newtonian flow: viscous stress force is proportional to velocity gradient:

$$\tau = \mu(\nabla \mathbf{u} + \nabla \mathbf{u}^T), \tag{2.8}$$

  where μ is dynamic viscosity of the fluid.

Consequently, for steady incompressible flow, Navier-Stokes' equations in integral form, can be expressed as:

$$\int_{S_{CV}} \rho \mathbf{u}\mathbf{u} \cdot \mathbf{n} dS = \int_{S_{CV}} \mu(\nabla \mathbf{u} + \nabla \mathbf{u}^T) \cdot \mathbf{n} dS + \int_{S_{CV}} (-\nabla p) \cdot \mathbf{n} dS \tag{2.9}$$

$$\int_{S_{CV}} \rho \mathbf{u} \cdot \mathbf{n} dS = 0 \tag{2.10}$$

where Eq. (2.9) is the momentum equation and Eq. (2.10) is the continuity equation.

GPDE is based on the Finite Volume Method (FVM), also known as control volume method. If the computation domain is subdivided into a finite number of small control volumes by a grid, managing the governing equations in integral form balanced in each finite volume (or control volume) will make the conservation satisfied for the whole system, in which way the Finite Volume Method is carried out. And

the discrete form of continuity and momentum equations of each finite volume can be expressed:

$$\sum_{N_{S_k}} \rho\phi_i \mathbf{u}_k \cdot \mathbf{n}\Delta S_k - \sum_{N_{S_k}} \mu\nabla\phi_i \cdot \mathbf{n}\Delta S_k = -\sum_{N_S} \nabla p \cdot \mathbf{n}\Delta S_k \tag{2.11}$$

$$\sum_{N_S} \rho\mathbf{u}_k \cdot \mathbf{n}\Delta S_k = 0 \tag{2.12}$$

where k is the index of the control volume face; ϕ_i is the mean value of velocity components in this control volume; $\mathbf{u}_k$ is the face velocity; p is the fluid static pressure; $\mathbf{n}$ is the unit face normal vector; ΔS_k is the face area; N_{S_k} is the number of volume faces.

FVM inherits the conservation theory of Reynolds' Transport Theorem and achieves it in form of discretization, which ensures flux balance of variables in each control volume. Accumulation of fluxes confirms mass and momentum conservation through the whole domain. Because of advantages of physical perspective and conservativeness, finite volume method is widely used in commercial CFD software, such as CFX/ANSYS, FLUENT and STAR-CD [90].

2.2 SIMPLE algorithm

For the discretized incompressible N-S equations, SIMPLE algorithm is employed to solve the system. The algorithm applied for decoupling velocity and pressure and updating them to convergence for incompressible steady system is tricky. In an incompressible flow system, pressure as a part of the source term is present in the momentum equation. There is no independent equation for pressure in governing equations, so iteration of pressure cannot be carried out directly. In addition, absolute pressure cannot be determined by the system, because only the gradient of pressure as constrain is involved in momentum equation. Therefore, an approach to derive an equation to solve the pressure on the basis of given momentum and continuity equations is crucial.

There are two levels of loops in SIMPLE, they are connected through the pseudo-time stepping. The first one refers to outer iteration that updates the pressure and velocity through a segregated manner. Each pseudo-time step is one outer iteration for the computation of velocity and pressure field. The other one is inter iteration which deal with the decoupled systems of pressure and velocity respectively.

Instead of using actual pressure value, SIMPLE algorithm proposed by Patanker and Spalding in 1972 [88]is applied to solve coupled problem of the velocity and the pressure via pressure and velocity correction approaches.

The discrete momentum equation can be expressed as:

$$A_I^{\phi_i}\phi_{i,I} + \sum_J A_J^{\phi_i}\phi_{i,J} = -(\frac{\delta p}{\delta x_i})_I \Delta\Omega \tag{2.13}$$

where I is the index of control volume I; J index denotes the neighbour control volume; A_I and A_J are the coefficients of unknown velocity components of ϕ_i at the center of cell I and J, which will be discussed in detail in later section; $\Delta\Omega$ is the volume of cell I. Through the pressure-correction approach, the velocity components ϕ_i and pressure p are taken as two parts values, including the value from last outer iteration and the correction part obtained at current iteration:

$$\phi_i^m = \phi_i^{m-1} + \phi^{'}, \quad p^m = p^{m-1} + p'. \tag{2.14}$$

According to governing system, the momentum equations should be satisfied at each outer iteration. Then introducing ϕ^{m-1}, ϕ^m,p^m and p^{m-1} into momentum equations Eq. (2.9), we can get:

$$A_I^{\phi_i}\phi_{i,I}^{m-1} + \sum_J A_J^{\phi_i}\phi_{i,J}^{m-1} = -(\frac{\delta p}{\delta x_i})_I^{m-1} \Delta\Omega \tag{2.15}$$

and

$$A_I^{\phi_i}\phi_{i,I}^{m} + \sum_J A_J^{\phi_i}\phi_{i,J}^{m} = -(\frac{\delta p}{\delta x_i})_I^{m} \Delta\Omega \tag{2.16}$$

Subtraction of Eq. (2.15) from Eq. (2.16) gives:

$$A_I^{\phi_i}\phi_{i,I}^{'} + \sum_J A_J^{\phi_i}\phi_{i,J}^{'} = -(\frac{\delta p^{'}}{\delta x_i})_I \Delta\Omega \tag{2.17}$$

Then the velocity correction equation can be obtained from above equation:

$$\phi_{i,I}^{'} = -\frac{\sum_J A_J^{\phi_i}\phi_{i,J}^{'}}{A_I^{\phi_i}} - \frac{1}{A_I^{\phi_i}}(\frac{\delta p^{'}}{\delta x_i})_I \Delta\Omega \tag{2.18}$$

If we ignore the influence of neighbour velocities, omitting the first term in above equation, we can get the relationship between velocity correction and pressure correction, which is the main approximation applied in SIMPLE algorithm:

$$\phi_{i,I}^{'} = -\frac{1}{A_I^{\phi_i}}(\frac{\delta p^{'}}{\delta x_i})_I \Delta\Omega \tag{2.19}$$

and then the corrected velocity is:

$$\phi_{i,I}^{m-1} + \phi_{i,I}^{'} = \phi_{i,I}^{m-1} - \frac{1}{A_I^{\phi_i}}(\frac{\delta p^{'}}{\delta x_i})_I \Delta\Omega \tag{2.20}$$

where the corrected velocity should also satisfy the continuity equation, so introduce this velocity to the discrete continuity equation to calculate the pressure correction $p^{'}$:

$$\frac{\delta}{\delta x_i}\left(\frac{1}{A_I^{\phi_i}}(\frac{\delta p^{'}}{\delta x_i})_I \Delta\Omega\right) = \frac{\delta}{\delta x_i}\left(\phi_{i,I}^{m-1}\right) \tag{2.21}$$

In general, the SIMPLE algorithm is implemented via the following steps:

1) Guess a pressure field p^* to build the right hand side (pressure source term) of momentum equation; and a guessing velocity field ϕ_i^* is used to create the coefficients A_I and A_J, which will be discussed in the discrete scheme section;

2) Solve the momentum equations Eq. (2.9) in order to get more accurate ϕ_i^{m*}, which doesn't satisfy the continuity equation;

3) Solve continuity equation Eq. (2.21) to get pressure correction $p^{'}$ based on velocity ϕ_i^{m*} obtained above;

4) Calculate velocity correction based on the pressure correction $p^{'}$ via Eq. (2.19);

5) Correct pressure and velocities by:

$$p^m = p^* + p', \quad \phi_i^m = \phi_i^{m*} + \phi_i^{'}, \tag{2.22}$$

5) Update velocity ϕ_i^* and pressure p^* in step 1) with current p^m and ϕ_i^m;

6) Repeat step 1) - step 5) until both momentum and continuity equations are satisfied. Meanwhile the correction values meets $p^{'} \leq \varepsilon$ and $\phi_i^{'} \leq \varepsilon$, where ε is small positive real number. It needs to notice that when whole system converges, the correction values are supposed to converge to zero.

2.3 Spatial discretion

GPDE applies co-located cell-centred data structure, which means variables are stored at cell centroid of grid cell and velocity components and pressure share the same location. For a two dimensional case shown in Fig 2.1, the grey cell is the control volume with cell centroid C_I. Grid cell is also control volume and grid nodes are control volume vertices. Abutting control volume $J1$ and I share the face k and

point M_k marks the face mid-point. Face normal vector $\mathbf{n}$ locates face midpoint. Arbitrary scalar variable ϕ_I standing for mean value of control volume I locates at cell centroid C_I. Although FVM ensures the conservation of the governing equations

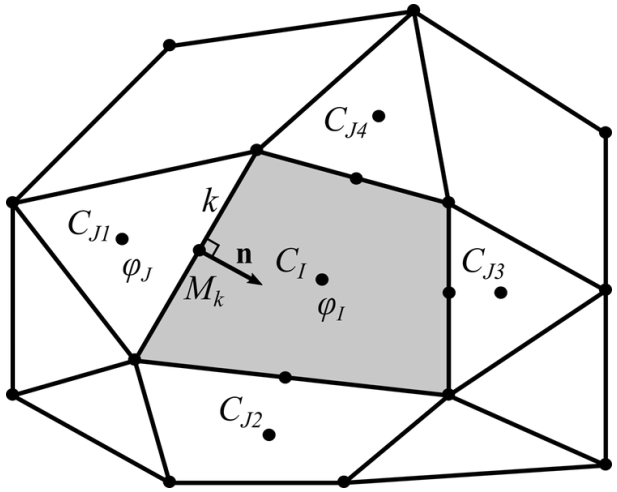


Figure 2.1: Values' location with cell-centred scheme

in the discretization form, how to calculate fluxes is very crucial to determine the accuracy of the modelling compared to the real physical problems. Fluxes are obtained using the values of variables at the volume face. Therefore approximations of each terms, including convection and diffusion in momentum equation, needs to be chose in order to build the coefficients of unknown values. Different flux construction methods used in GPDE are discussed in the following sections.

2.3.1 Upwind Differencing Scheme (UDS)

The core point of upwind differencing scheme (UDS) is that we take information from the upstream to build the discretization form of first order derivative. In UDS, the derivative is expressed as:

$$\phi_{i,k} = \begin{cases} \phi_{i,I} & for, (\mathbf{u}\cdot\mathbf{n})_k > 0 \\ \phi_{i,J} & for, (\mathbf{u}\cdot\mathbf{n})_k < 0 \end{cases} \tag{2.23}$$

where $\phi_{i,I}$ is the upwind value of velocity components at cell I volume center; $\phi_{i,J}$ is the value of velocity components at neighbour cell J volume center, which is also the upwind value, if the face velocity $(\mathbf{u}\cdot\mathbf{n})_k < 0$. It is well known that UDS with 1st order accuracy is important to be applied in convection term in order to obtain numerically stable results.

2.3.2 Central Differencing Scheme (CDS)

Comparing with UDS, central differencing scheme (CDS) includes both information of upstream and downstream, which has 2nd order accuracy. In CDS, this approximation can be obtained by the average of values from adjacent volumes:

$$\phi_{i,k} = \frac{1}{2}(\phi_{i,I} + \phi_{i,J}) \tag{2.24}$$

For non-uniform grid, the variables at face have different distances to the variables at centroids of volume I and J. Then the influence of both sides of centroid values needs to be modified by the geometric weight:

$$\phi_{i,k} = \lambda_k \phi_{i,I} + (1 - \lambda_k)\phi_{i,J} \tag{2.25}$$

where λ_k as a geometric weight is defined in this way:

$$\lambda_k = \frac{(\mathbf{r}_M - \mathbf{r}_I) \cdot \mathbf{n}}{(\mathbf{r}_M - \mathbf{r}_J) \cdot \mathbf{n}} \tag{2.26}$$

where M is the midpoint of the face; $\mathbf{r}_M$, $\mathbf{r}_I$ and $\mathbf{r}_J$ are displace vectors of M, I and J; $\mathbf{n}$ is normal vector of the face between cell I and J.

In GPDE CDS is also used for modifying coefficients of face value, such as $\frac{1}{A^{\phi_n}}$ in Eq. (2.46):

$$\left(\frac{1}{A^{\phi_n}}\right)_k = \lambda_k \left(\frac{1}{A^{\phi}}\right)_I + (1 - \lambda_k) \left(\frac{1}{A^{\phi}}\right)_J \tag{2.27}$$

where the face value can get more accurate approximation.

2.3.3 Evaluation of Gradients

Spatial gradients of the flow field are needed for the calculation of the higher-order convective fluxes and for the viscous fluxes. According to Taylor series expansion, more accurate approximations can be constructed by adding the correction using the gradient. Consider the calculation of face value for example. For face k, the face value on the cell I side can be modified as:

$$\phi_k^+ = \phi_I + \nabla\phi_I \cdot (\boldsymbol{r}_I - \boldsymbol{r}_M) \tag{2.28}$$

Similarly, the face value on the cell J side can be modified as:

$$\phi_k^- = \phi_J + \nabla\phi_J \cdot (\boldsymbol{r}_J - \boldsymbol{r}_M) \tag{2.29}$$

Then the face value could take the average of ϕ_k^+ and ϕ_k^- rather than $\phi_{k,I}$ and $\phi_{k,J}$.

There are two popular approaches to compute gradients, the Green-Gauss and the Least-Squares approach [91]. Both have advantages and disadvantages, as discussed below:

- Green-Gauss Approach
  All the vector field can be treated as several scalar fields along each coordinate direction as discussed above. For a scalar function ϕ_I, e.g. the velocity component in a certain direction, varying over an arbitrary cell I, the gradient can be obtained from the integral of all the ϕ value given on the every face between the cell I and other adjacent cells J ($J = 1, 2, ..., N_F$):

$$\nabla\phi \approx \frac{1}{\Omega}\sum_{J=1}^{N_F}\frac{1}{2}(\phi_I + \phi_J)\overrightarrow{n}_k\Delta S_k \tag{2.30}$$

  where N_F is the number of neighbouring cells.

  If the the non-orthogonality of grid is severe, which often occurs on unstructured grids, the average value in Eq. (2.30) is not accurate because the face value is not the median of abutting cells'. In GPDE, the weighted average method is applied when the modified value at the face between ϕ_I and ϕ_J is pursued, then the gradient of ϕ in cell I is:

$$\nabla\phi_I \approx \frac{1}{\Omega}\sum_{J=1}^{N_F}[\lambda_k\phi_I + (1-\lambda_k)\phi_J]\overrightarrow{n}_k\Delta S_k \tag{2.31}$$

  where the definition of λ_k is the same with Eq. (2.26):

- Least-Squares Approach
  The Least-Squares approach is based on the first-order Taylor series approximation. For the variation of a scalar ϕ_I in a cell I with the gradient $\nabla\phi_I$

$$\nabla\phi_I = \begin{bmatrix} \dfrac{\partial\phi}{\partial x} \\ \dfrac{\partial\phi}{\partial y} \end{bmatrix}_I \tag{2.32}$$

  In the direction of the normal to each face, the projected gradient should approximate the difference in values along the edge $J - I$

$$(\nabla\phi_I)\cdot\overrightarrow{r}_{IJ} = \phi_J - \phi_I \tag{2.33}$$

with the vector $\overrightarrow{r}_{IJ}$ between cell centres i, j as

$$\overrightarrow{r}_{IJ} = \begin{bmatrix} \Delta x_1 \ \Delta y_1 \\ \Delta x_2 \ \Delta y_2 \\ \vdots \ \vdots \\ \Delta x_{N_F} \ \Delta y_{N_F} \end{bmatrix} \tag{2.34}$$

In a linearly varying field, the first-order approximation in Eq. (2.33) must hold for all edges connected to cell I:

$$\begin{bmatrix} \Delta x_1 \ \Delta y_1 \\ \Delta x_2 \ \Delta y_2 \\ \vdots \ \vdots \\ \Delta x_{N_F} \ \Delta y_{N_F} \end{bmatrix} \begin{bmatrix} \partial_x \phi \\ \partial_y \phi \end{bmatrix}_I = \begin{bmatrix} \phi_1 - \phi_I \\ \phi_2 - \phi_I \\ \vdots \\ \phi_{N_F} - \phi_I \end{bmatrix}. \tag{2.35}$$

However, the overdetermined(over-constrained) equation system will be unsolvable for variations other than linear. The system Eq. (2.35) is written as

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{2.36}$$

The least-square approach tries to minimize the residual $\mathbf{R} = \mathbf{b} - \mathbf{A}\mathbf{x}$ rather than solving the equation exactly. Two-norm of the residual is minimised in least squares problems. From a given $\mathbf{x}$, the directional derivative in any direction δx is

$$\begin{aligned} \nabla_x |\mathbf{R}|^2 \cdot \delta x &= 2\langle \mathbf{A}\delta x, \mathbf{b} - \mathbf{A}\mathbf{x} \rangle \\ &= 2\delta x^T \left(\mathbf{A}^T \mathbf{b} - \mathbf{A}^T \mathbf{A}\mathbf{x} \right). \end{aligned} \tag{2.37}$$

When all possible directional derivatives are zero, the two-norm achieves the minimum. And then we obtain

$$\mathbf{A}^T \mathbf{A}\mathbf{x} = \mathbf{A}^T \mathbf{b} \tag{2.38}$$

which is called the normal equations.[2] Similarly, the full rank equations are obtained via both sides of Eq. (2.35) multiplying by the transposed matrix $\mathbf{A}^T$. It reduces the potential rank of the equations' system, and a $N \times N$ coefficient matrix is a reasonable choice of our modification direction.

Comparing these two approaches, we can draw a safe conclusion that the least-square approach needs to calculate the equation system, which requires more com-

[2]Normal equations specify that the residual must be normal (orthogonal) to every vector in the span of $\mathbf{A}$[92].

putation than the Green-Gauss approach. Additionally, experience shows that the gradient calculated via Green-Gauss approach on mixed grid will become highly inaccurate[91].

2.3.4 Deferred-Correction approaches

In order to reduce the size of the computational molecule (for accelerating the speed of computation), only the contributions from the nearest (first-order) neighbours J connected to node I by a face are kept on the left hand side in the matrix $\mathbf{A}$. However, this simple approximation is usually not accurate enough and higher order approximations to the flux are needed it involves contributions from second-order and further neighbours, such as the gradient evaluation for higher accuracy. A compact stencil for a small matrix dictates that all the calculation from higher-order neighbours are kept on the right-hand-side. The fluxes restruction approach implemented in GPDE is deferred-correction: put the higher-order approximation on the right hand side; put the simpler approximation both on the left (unknown variables) and the right sides (the variables calculated from the last iteration) [93]. Then the right hand side has a term of the difference between two types of approximations of the same term, which is likely to be small (as the correction part of right hand side). For example, in FVM the flux passing through a face can be expressed:

$$F_k = \underbrace{F_k^L}_{\text{left hand side}} + \underbrace{(F_k^H - F_k^L)^{m-1}}_{\text{right hand side}} \tag{2.39}$$

where superscript $m-1$ is the index of last iteration; L and H denote 'low' and 'high' accuracy.

Therefore, to generate the coefficient matrix in GPDE, all the terms' (including convective and diffusive) approximations are corrected via deferred-correction approach.

- Convective term

  1. The first order upwind scheme is applied to convective flux:

     $$F_k^c = max(\dot{f}, 0)\phi_I + min(\dot{f}, 0)\phi_J \tag{2.40}$$

     where $\dot{f} = \mathbf{u}_k \cdot \mathbf{n}$ is the mass flux at face k. Then we can generate the simplest approximation in coefficient matrix in order to reduce the size of computational molecule and to prevent divergence as well.

  2. Higher order schemes can be used as an option by adding the correction

term on the right hand side:

$$F_{cor} = CBL * (F^{imp} - F^{exp}) \tag{2.41}$$

where F^{imp} is approximated from UDS (1st order accuracy) and F^{exp} can be chosen from CDS, Min-Mod, Yan-Leer and Superbee schemes (2nd order accuracy) from TVD schemes class[90]. And CBL stands for the coefficient of convection blending scheme , which, from 0 to 1, can be used to regulate the degree of mixing different schemes. When $CBL = 0$, only UDS plays the role and it is most stable; when $CBL = 1$, the highest accuracy could be achieved, which may result in a scheme with lower stability.

- Diffusive term

1. The simplest approximation for diffusive term - Central Difference Scheme (CDS) is adopted for diffusive term to generate the coefficient matrix (left side of equations):

$$F_k^d = -D \cdot (\phi_J - \phi_I) \tag{2.42}$$

where the diffusion coefficient D (including the viscosity information) is constant obtained from weighted average value according to the distances between cell centroids and midpoint at the face:

$$D = [\lambda_k \mu_I + (1 - \lambda_k)\mu_J] \frac{\Delta S_k}{\Delta l_{IJ}} \tag{2.43}$$

where ΔS_k is the face area; Δl_{IJ} is the distance between two cell-centroids of cell I and J; λ_k is the face geometric weight.

2. For the diffusive term, the higher order approximation should have been to use second order central difference (CDS) value of ϕ instead of first order upwind differencing scheme (UDS).

   The correction term is built via geometric modification. The implicit part of diffusion is based on the difference between cell centroids, so the gradient of ϕ should be corrected to the face normal direction. From this view, in GPDE, for diffusive correction term, the *Muzaferija* method[93] is applied, in which the diffusion correction is expressed as:

$$F_{cor} = \mu_k \Delta S_k \nabla \phi_k \cdot (\boldsymbol{n} - \boldsymbol{i}_\xi) \tag{2.44}$$

where $\boldsymbol{i}_\xi$ is the direction of the line connecting two adjacent cell centroids.

Face gradient $\nabla\phi_k$ is the linear interpolated gradient:

$$\nabla\phi_k = \lambda_k \nabla\phi_I + (1-\lambda_k)\nabla\phi_J \tag{2.45}$$

2.3.5 Pressure correction

Simply discretizing pressure gradient by central difference scheme leads to check-board pressure layout, where staggered grid is widely used to achieve reasonable pressure field for structured grid. However, for unstructured grid computation, the collocated arrangement is more conveniently adopted for complex geometry. In GPDE, as it is mentioned above that all the variables velocity component ϕ_i and pressure p are located at the centroid of the control volume. In order to avoid getting check-board pressure field, the pressure calculation needs to be treated carefully.

Considering an arbitrary face k, the face velocity comes from the interpolated value of cell center, which needs to be corrected by subtracting the difference between the pressure gradient and the interpolated gradient at the cell face location:

$$\phi_{i,k}^{m*} = \overline{(\phi_i^{m*})}_k - \Delta\Omega_k \overline{\left(\frac{1}{A_I^{\phi_i}}\right)}_k \left[\left(\frac{\delta p}{\delta x_i}\right)_k - \overline{\left(\frac{\delta p}{\delta x_i}\right)}_k\right]^{m-1} \tag{2.46}$$

where $\phi_{i,k}^{m*}$ is the corrected velocity component value at face k, $\overline{(\phi_i^{m*})}_k$ is the interpolated velocity component at cell center, $\left(\frac{\delta p}{\delta x_i}\right)_k$ is the gradient of pressure, and $\overline{\left(\frac{\delta p}{\delta x_i}\right)}_k$ is interpolated gradient. This approach also called as momentum interpolated method is proposed by Rhie and Chow (1983) [93].

All the coefficients of velocity components are the same at the cell centroids, but the face value we need to correct it by using the weighted average value of $A_I^{\phi_i}$ from both sides of the face to replace that coefficient of velocity in the face normal direction:

$$\phi_{n,k}^{m*} = \overline{(\phi_n^{m*})}_k - \Delta\Omega_k \overline{\left(\frac{1}{A^{\phi_n}}\right)}_k \left[\left(\frac{\delta p}{\delta n}\right)_k - \overline{\left(\frac{\delta p}{\delta n}\right)}_k\right]^{m-1} \tag{2.47}$$

Based on this approach, the continuity equation is built in a modified way:

1. Calculate the mass fluxes passing through a control volume using interpolated velocity component:

$$\Delta\dot{f} = \sum_k \rho\overline{(\phi_n^{m*})}_k \Delta S_k \tag{2.48}$$

   where $\Delta\dot{f}$ is the change mass flow of the control volume, which equals to the mass fluxes through all the control volume faces; ΔS_k is the area of face k.

2. The mass fluxes above obtained from momentum equation can not satisfy the

continuity, which needs to be corrected by the fluxes change at faces using the velocity correction:

$$\dot{f}'_k = \rho\phi'_{n,k}\Delta S_k \tag{2.49}$$

where the velocity correction can be obtained via Eq. (2.19), and then the equation is expressed as:

$$\dot{f}'_k = \rho u'_{n,k}\Delta S_k = -(\rho\Delta\Omega\Delta S)_k\overline{\left(\frac{1}{A_P^{\phi_n}}\right)}_k\left(\frac{\delta p'}{\delta n}\right)_k \tag{2.50}$$

3. The corrected mass fluxes satisfy the continuity equation:

$$\sum \dot{f}'_k + \Delta\dot{f} = 0 \tag{2.51}$$

where we obtain the pressure correction equation.

All the coefficients of velocity components are the same at the cell centroids, but in pressure correction equation, the coefficients reciprocal values $\frac{1}{A_I^{\phi_i}}$ needs to be switched to face values $(\frac{1}{A_I^{\phi_i}})_k$. Therefore, the face value is interpolated by using the weighted average value of $\frac{1}{A_I^{\phi_i}}$ from both sides of the face to replace that coefficient of velocity in the face normal direction:

$$\sum_k \dot{f}'_k = -\sum_k\left\{\overline{(\phi_n^{m*})}_k - \Delta\Omega_k\overline{\left(\frac{1}{A^{\phi_n}}\right)}_k\left[\left(\frac{\delta p}{\delta n}\right)_k - \overline{\left(\frac{\delta p}{\delta n}\right)}_k\right]^{m-1}\right\} \tag{2.52}$$

where $\dot{f}'_k$ is expressed via pressure correction p':

$$\dot{f}'_k = \rho u'_{n,k}\Delta S_k = -(\rho\Delta\Omega\Delta S)_k\overline{\left(\frac{1}{A_P^{\phi_n}}\right)}_k\left(\frac{\delta p'}{\delta n}\right)_k \tag{2.53}$$

Then pressure correction equation is expressed as:

$$\begin{aligned}&\sum_k -(\rho\Delta\Omega\Delta S)_k\overline{\left(\frac{1}{A_P^{\phi_n}}\right)}_k\left(\frac{\delta p'}{\delta n}\right)_k \\ &= -\sum_k\left\{\overline{(\phi_n^{m*})}_k - \Delta\Omega_k\overline{\left(\frac{1}{A^{\phi_n}}\right)}_k\left[\left(\frac{\delta p}{\delta n}\right)_k - \overline{\left(\frac{\delta p}{\delta n}\right)}_k\right]^{m-1}\right\}.\end{aligned} \tag{2.54}$$

In pressure correction equation Eq. (2.54), the face normal gradient is discretilised via CDS:

$$\left(\frac{\delta p'}{\delta n}\right)_k \approx \frac{p'_J - p'_I}{l_{IJ}} \tag{2.55}$$

where l_{IJ} is the distance from CV I cell centroid to CV J cell centroid and face normal vector is assumed along the direction from I to J. For CV I the accounted flux of its neighbours mean value p'_J is:

$$\dot{f} = -(\rho\Delta\Omega\Delta S)_k \overline{\left(\frac{1}{A_P^{\phi_n}}\right)}_k \frac{p'_J}{l_{IJ}} \tag{2.56}$$

and then coefficient value for p'_J is

$$\begin{aligned} D_p &= -(\rho\Delta\Omega\Delta S)_k \overline{\left(\frac{1}{A_P^{\phi_n}}\right)}_k \frac{1}{l_{IJ}} \\ &= -\rho \overline{\left(\frac{1}{A_P^{\phi_n}}\right)}_k \Delta S_k^2 \end{aligned} \tag{2.57}$$

Similarly, for CV I's neighbour CV J, negative flux is accounted in order to guarantee flux conservation:

$$\dot{f} = -(\rho\Delta\Omega\Delta S)_k \overline{\left(\frac{1}{A_P^{\phi_n}}\right)}_k \frac{p'_I}{l_{IJ}} \tag{2.58}$$

so the coefficient value for p'_I is

$$\begin{aligned} D_p &= -(\rho\Delta\Omega\Delta S)_k \overline{\left(\frac{1}{A_P^{\phi_n}}\right)}_k \frac{1}{l_{IJ}} \\ &= -\rho \overline{\left(\frac{1}{A_P^{\phi_n}}\right)}_k \Delta S_k^2 \end{aligned} \tag{2.59}$$

where it can be found that the same value of coefficient comes from neighbour cells' contribution.

2.4 Boundary conditions

The unique solution of discretized algebraic system is determined by well-imposed boundary conditions. How to set boundary conditions is an essential and difficult issue in CFD problems. For incompressible Navier-Stokes equations, prescribed velocity boundary and pressure boundary constrain flow to a converged fields' distribution. Boundary conditions are specified according to the physical model of practical problems to be solved. On the other hand, the adopted boundary conditions should satisfy numerical requirement as well, because ill-imposed boundary condition will not bring reasonable unique solution and may diverge the system. Therefore, the boundary conditions are critical components of flow solver. Normally, there are stan-

dard boundary conditions widely-applied for majority NS computations, which are implemented in GPDE.

In momentum equation, velocity as primitive variable is computed. The unknown variable ϕ_i is velocity component. Then for velocity vector $\mathbf{u}(u, v, w)$, the scalar $\phi_1 = u$,$\phi_2 = v$ and $\phi_3 = w$ are computed separately but sharing the same coefficient matrix $\mathbf{A}$. Cell I is a control volume at a boundary, which is the grey area shown in Fig. 2.2. CV I's mean velocity and its components' scalar are $\mathbf{u}_I$ and ϕ_I. The grey bottom line is a physical boundary. Boundary velocity vector locates face midpoint is $\mathbf{u}_b$ and scalar boundary value is ϕ_b. Boundary face normal vector and tangent vector are $\mathbf{n}_b$ and τ_b.

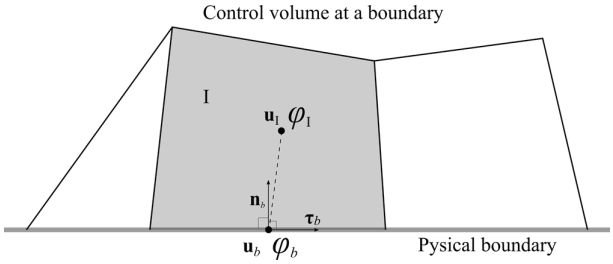


Figure 2.2: Control volume at a physical boundary

1. Inlet boundary

   For given inlet mass flow cases, velocity is fixed at inlet. The 1st type (or Dirichlet) boundary condition is implemented for constant value boundary condition in momentum linear equations. Explicit inlet boundary mass flow fluxes is formulated as:

   $$\mathbf{u}_b = \mathbf{u}_{in} \tag{2.60}$$

   And once inlet velocity is set, it doesn't need to be updated or corrected during entire computation.

2. Outlet boundary

   Fully developed outlet is always assumed, where outlet velocity is dependent on up stream flow. Zero gradient of velocity at outlet is assumed. The 2nd (Neumann) boundary condition is applied for outlet boundary velocity:

   $$\frac{\partial \mathbf{u}_b}{\partial n_b} = 0 \tag{2.61}$$

   For 1st order accuracy, outlet velocity equals upstream CV's velocity:

   $$\mathbf{u}_b = \mathbf{u}_I; \tag{2.62}$$

For 2nd order accuracy, outlet velocity equals extrapolated value from CV I:

$$\mathbf{u}_b = \mathbf{u}_I + \nabla \mathbf{u}_I \cdot \mathbf{r} \tag{2.63}$$

In order to keep a small stencil when building up the linear system, 1st order formulation is applied for equations' coefficient matrix; 2nd order accuracy is achieved via post explicit correction for boundary velocity after momentum linear system computation.

3. Wall boundary

   For impermeable wall, non-slip wall is assumed, and zero velocity is given at wall boundary. It is also the 1st type (or Dirichlet) boundary condition in mathematics.

$$\mathbf{u}_b = \mathbf{u}_{wall} = 0 \tag{2.64}$$

4. Symmetric boundary

   Velocities at two sides of symmetric boundary are identical. Zero gradient of velocity is also assumed here. Different from outlet boundary condition, there is no convection through symmetric boundary, so the velocity at symmetric boundary is along the face tangent direction.

$$\mathbf{u}_n = 0, \mathbf{u}_b = \mathbf{u}_\tau \tag{2.65}$$

   For 1st order accuracy, boundary velocity equals tangent component of cell centroid's value:

$$\mathbf{u}_b = \mathbf{u}_I - \mathbf{u}_I \cdot \mathbf{n}_b \tag{2.66}$$

   For 2nd order accuracy, extrapolating velocity from cell centroid's one via gradient evaluation:

$$\mathbf{u}_{tmp} = \mathbf{u}_I + \nabla \mathbf{u}_I \cdot \mathbf{r} \tag{2.67}$$

   and then project this intermediate value to boundary tangent direction:

$$\mathbf{u}_b = \mathbf{u}_{tmp} - \mathbf{u}_{tmp} \cdot \mathbf{n}_b \tag{2.68}$$

In SIMPLE algorithm, pressure is composed of updated pressure p^* and pressure correction $p^{'}$.

$$p^{new} = p^* + p^{'} \tag{2.69}$$

After momentum equation, pressure correction value is obtained from continuity equation rather than absolute pressure. Only pressure correction part comes from

linear algebraic system. Implementation of pressure boundary condition needs to considerate both of components in inner linear equation and outer iteration updating. Physically, when it come with flat wall boundary, pressure should satisfy the zero gradient condition:

$$\frac{\partial p}{\partial n} = 0, \tag{2.70}$$

which leads to:

$$\frac{\partial (p^* + p^{'})}{\partial n} = \frac{\partial p^*}{\partial n} + \frac{\partial p^{'}}{\partial n} = 0. \tag{2.71}$$

This condition can be achieved via:

$$\frac{\partial p^*}{\partial n} = 0 \qquad \text{and} \qquad \frac{\partial p^{'}}{\partial n} = 0 \tag{2.72}$$

At the inlet, when inlet velocity is prescribed, the mass flux correction in pressure correction equation is determined as zero for incompressible flow. Neumann boundary condition (zero gradient) should be satisfied.

At outlet, when outflow boundary is already far from the region of interest, the mass flow is normally extrapolated from upstream, which means mass flux correction is zero as well. Moreover, when flow is complex, back flow could happen to destroy mass conservation. Therefore, for pressure correction equation, the right hand side at the outlet boundary needs to be modified with the same inlet mass flow.

However, as it is mentioned above, all boundary should be Neumann boundary (zero gradient) in pressure correction equation, which would leads to a singular coefficient matrix. Normally, for incompressible flow, inlet pressure is set to be zero as reference to guarantee unique solution. So at the inlet, the pressure correction value is fixed as zero $p' = 0$.

1. Wall Boundary

   Zero gradient $\frac{\partial p'}{\partial n} = 0$ Neumann boundary condition is applied for flat wall:

   $$\left(\frac{\partial p'}{\partial n}\right)_b \approx \frac{p'_b - p'_I}{l_{MI}} = 0 \tag{2.73}$$

   which means there is no left hand side left, forcing coefficient of pole value $d_{ij} = 0$ (it should not be 1st order, if face normal gradient value is directly substituted to p' equation). Because face normal velocity on wall is zero, the right hand side (mass flux) is also zero, $c = 0$ and $rhs = rhs + c$ for wall boundary;

2. Inlet

   Because of prescribed velocity values at inlet, pressure doesn't need to be cor-

rected, which leads pressure correction value $p' = 0$ on inlet boundary:

$$\left(\frac{\partial p'}{\partial n}\right)_b \approx \frac{p'_b - p'_I}{l_{MI}} = \frac{0 - p'_I}{l_{MI}} \tag{2.74}$$

where l_{MI} is the distance from cell centroid I to face center M. Consequently, coefficient of pole value D is the same with interior cells':

$$D = -\left(\frac{1}{A_P^{\phi_n}}\right)_b \Delta S_b^2 \tag{2.75}$$

And right hand side:

$$RHS = -\dot{f} = -\rho u \Delta S_b \tag{2.76}$$

where f is local mass flux through boundary.

3. Outlet

   To guarantee mass conservation, mass flow at outlet is updated via inlet mass flow. Therefore, for p' equation, the outlet boundary local mass flow for pressure correction source needs to be modified via:

$$\dot{f}_b = \rho u_b \frac{\dot{f}_{inlet}}{\dot{f}_{outlet}} \Delta S_b \tag{2.77}$$

2.5 Linear solvers

Applying SIMPLE algorithm, N-S equations has been decoupled into two linear equations: discretised momentum and pressure correction equations. For solving linear algebraic equations, direct method and iterative method are two main types of methods. Due to the high cost of direct methods, e.g. Gaussian elimination and LU decomposition methods, iterative methods are frequently adopted with its efficiency for solving flow field, especially for computation on a large number of nodes. Meanwhile, in practical computation, error induced from the discrete schemes is usually larger than the accuracy of computer arithmetic[93] so there is no reason to solve the discrete equations with absolute accuracy. In addition, N-S equations are non-linear systems and pseudo-time stepping is applied for non-linear convergence of whole system. But linear inner equations are unnecessary to highly converge since the velocity and pressure values are corrected at every outer iteration. Numerical experience shows that smoothing the fields to 10^{-2}-10^{-3} relative residual reduction is enough for outer iteration's convergence of global residual. These two types of iterations' relationship is shown in Fig. 2.3. Therefore, iterative linear solver is preferred to apply for linear equations in SIMPLE algorithm. Because of upwind-

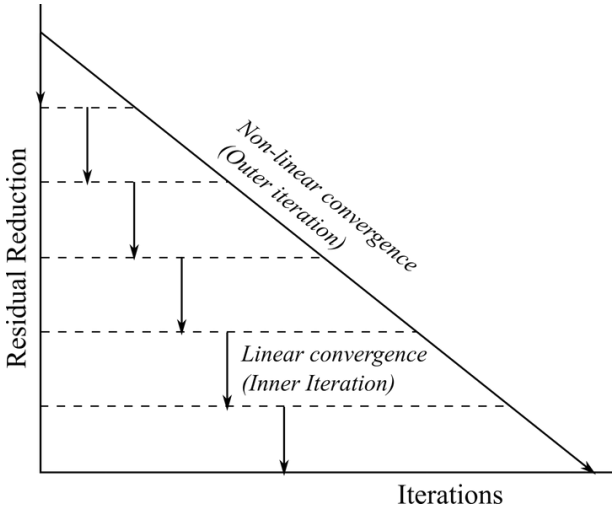


Figure 2.3: Outer iteration and inner iteration

bias scheme employed for convection term in momentum equation, flux contribution accounted for two neighbour CV is not identical, so the coefficient matrix is asymmetric matrix. Symmetric coefficient matrix is obtained from pressure correction equation. Because of the property of momentum and pressure correction equations, two iterative linear solvers are linked for solving respective linear system. In GPDE, conjugate gradient stabilized method (CGSTAB) and bi-conjugate gradient stabilized method [94] are respectively applied for solving continuity (pressure correction) equations and momentum equations. CGSTAB is a stabilised version of conjugate gradient solver for symmetric system and BI-CGSTAB is derived version of asymmetric system. Compared with classical iterations: Jocobi, Gauss-Seidel, successive over-relaxation (SOR) and so on, CGSTAB and BICGSTAB involve optimisation over Krylov spaces[95], which are widely used for solving sparse linear system in N-S equations.

2.6 Flow solver validation

2.6.1 Bench mark lid-driven cavity case

The "lid-driven cavity flow" is a classic test case which is often used for the validation of incompressible flow solvers [96]. This case is defined as follows:

- Unit square cavity in 2D: 1.0×1.0
- Top: moving lid: $u = 1.0, v = 0.0$;
- Other walls: $u = v = 0.0$ (non-slip boundary conditions);

Results for different Reynolds numbers are published in the literature [96]; the convergence and accuracy of the solver are validated against these for Re=100, 400

and 1000. A non-uniform 130×130 grid is used, as shown in Fig. 2.4 (left). In order to minimise the influence of the velocity singularity at the top corners, the mesh is refined at the corners (see details in Fig. 2.4 (right). Because the error mainly comes

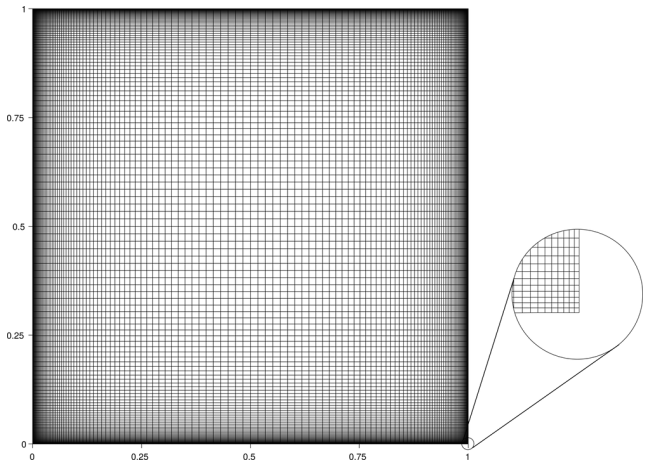


Figure 2.4: Grid adopted for cavity case

from the approximation of discrete schemes error, which is dependent on mesh size, refining the mesh near the singularities will provide more accurate results. According to the data from velocity field calculated by GPDE, the stream function satisfies:

$$u = \frac{\partial \psi}{\partial y}, v = -\frac{\partial \psi}{\partial x} \tag{2.78}$$

where u and v are two dimensional velocity components. Then the line integral of velocity component is the stream function, which is used to plot the streamline:

$$\psi = \int (udy - vdx) \tag{2.79}$$

If we choose the y direction as the integral direction, then the stream function is obtained via:

$$\psi = \int_{y_0}^{y_1} u(y)dy = \sum (u_I \Delta y) \tag{2.80}$$

Comparisons of streamlines between reference solution [96] and the GPDE solution at various Reynolds numbers shown in Figs. 2.5, 2.6 and 2.7 shows good agreement. The quantitative comparison of primary vortex locations is shown in Tab. 2.1.

Table 2.1: Primary vortex location of the lid-driven cavity case

Reynolds Number	Re=100	Re=400	Re=1000
Primary vortex location(x,y) in Ghia[96]	(0.6172,0.7344)	(0.5547,0.6055)	(0.5313,0.5625)
Primary vortex location(x,y) in GPDE	(0.6100,0.7448)	(0.5599,0.6100)	(0.5258,0.5599)

The error from integral calculation of velocity influences the results above to a certain extent, but they are still acceptable according to the streamline layout of the whole field. Through the qualitative comparison of locations of major and secondary vortices between Ghia (a) and GPDE (b), GPDE is reliable for simulation of two dimensional flow problems. Quantitative results are compared with the ones

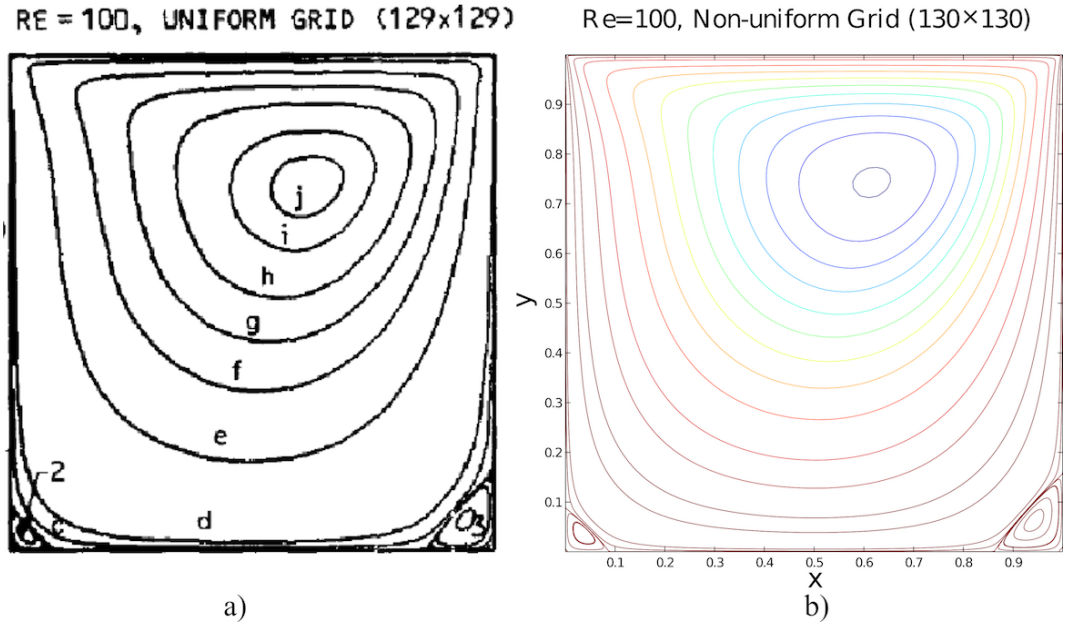


Figure 2.5: Streamlines comparison between a) Ghia[96] and b) GPDE of the lid-driven cavity flow at Re=100

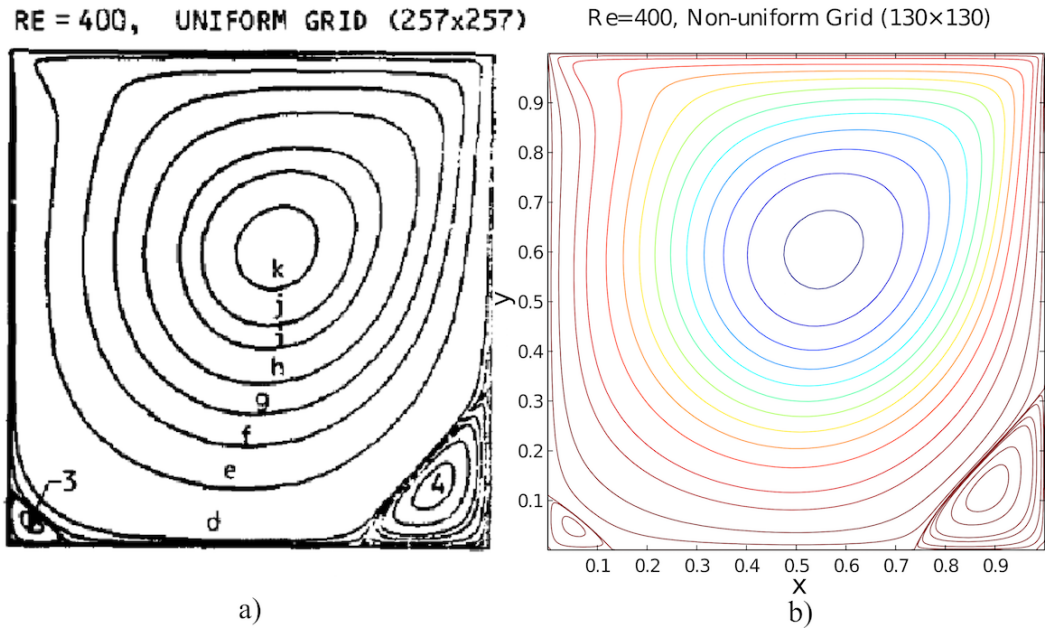


Figure 2.6: Streamlines comparison between a) Ghia[96] and b) GPDE of the lid-driven cavity flow at Re=400

provided by Ghia [96] as well, which are shown in Fig. 2.8, Fig. 2.9 and Fig. 2.10.

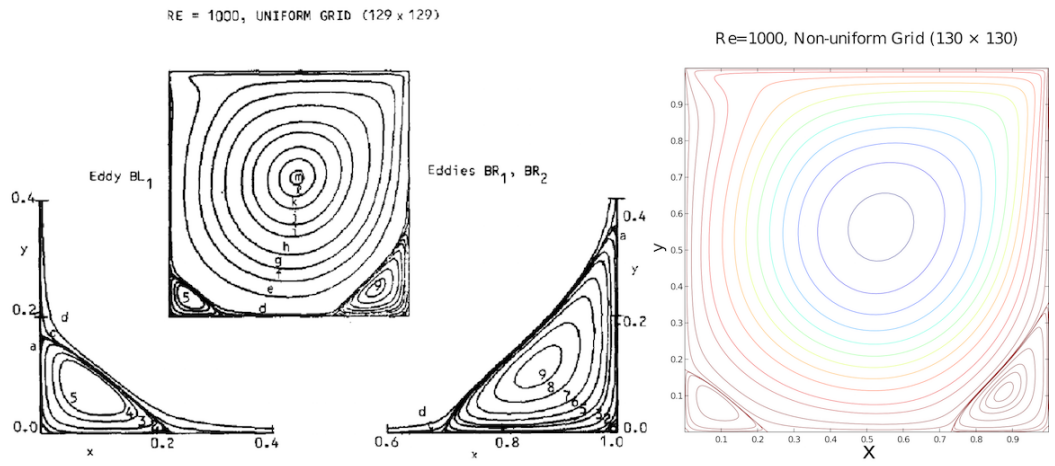


Figure 2.7: Streamlines comparison between a) Ghia[96] and b) GPDE of the lid-driven cavity flow at Re=1000

During the numerical experiments, it is found that for lower Re number, using less correction in convection (2.41) converge more quickly to the same accurate results by using a smaller CBL number (the range from 0.0 to 0.5). For higher Reynold (e.g. $Re = 1000$ for cavity case) number flow, the convection blending scheme coefficient CBL number needs to be chosen closer to/as 1.0 to converge comparatively more quickly, which means using higher schemes to correct the result let the residual reduce more quickly, and accelerate the convergence for fewer iterations when the flow problem is at high Reynold number.

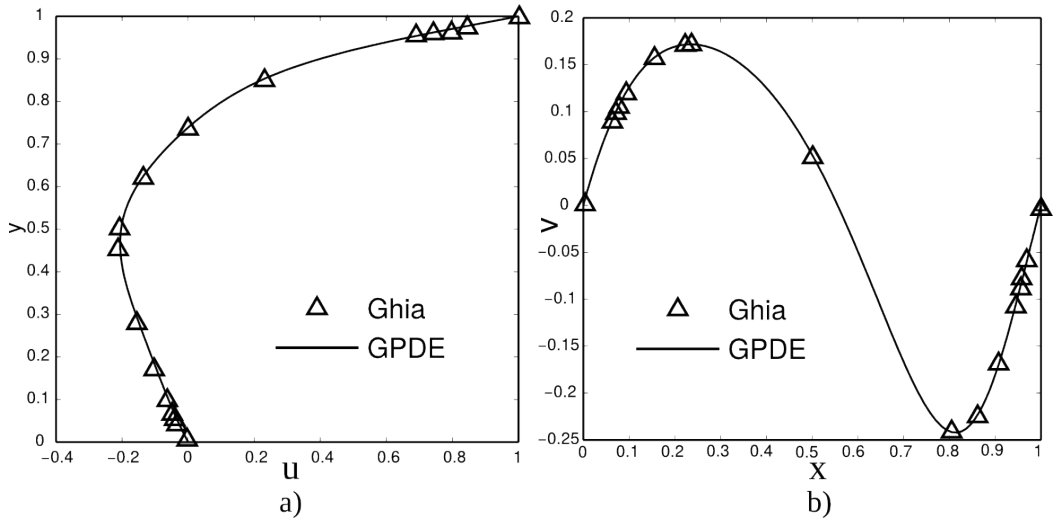


Figure 2.8: Re=100, results for u-velocity (a) and v-velocity (b) through the geometric center of cavity

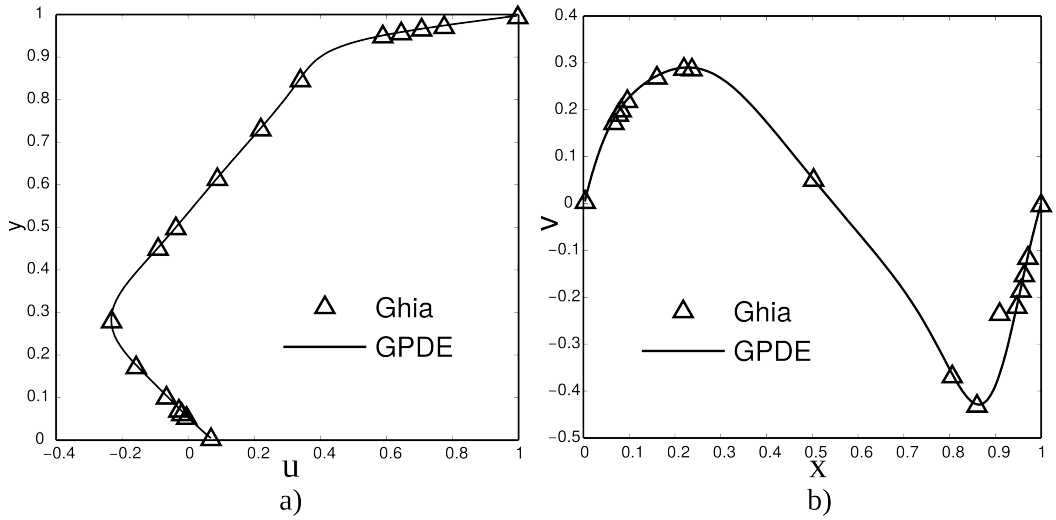


Figure 2.9: Re=400, results for u-velocity (a) and v-velocity (b) through the geometric center of cavity

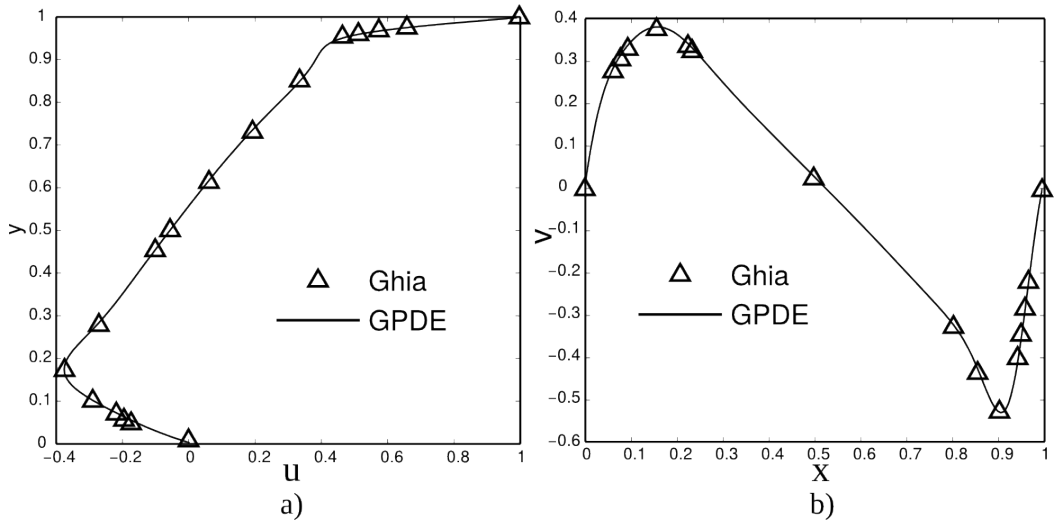


Figure 2.10: Re=1000, results for u-velocity (a) and v-velocity (b) through the geometric center of cavity

2.6.2 3D case compared with OpenFOAM

For 3 dimensional flow, the validation is carried out via comparing the solution with widely used open-source CFD software OpenFOAM. The test case is s-bend channel flow at Re=600 on 47k grid, shown in Fig. 2.11. Uniform inlet velocity is given, and outlet is assumed as fully developed boundary. The channel wall is non-slip wall boundary. For GPDE and OpenFOAM, SIMPLE algorithm is used, and flux scheme takes 1st order upwind scheme. The same inner tolerance is applied for both solvers. The major parameter settings are shown in Tab. 2.2. The converged solution is obtained at global residual reduced to 10^{-10}. This quantitative velocity values are chose over the same line in flow solution, shown in Fig 2.13. The agreement of solutions can be achieved via GPDE, which indicates the validated solver

implementation.

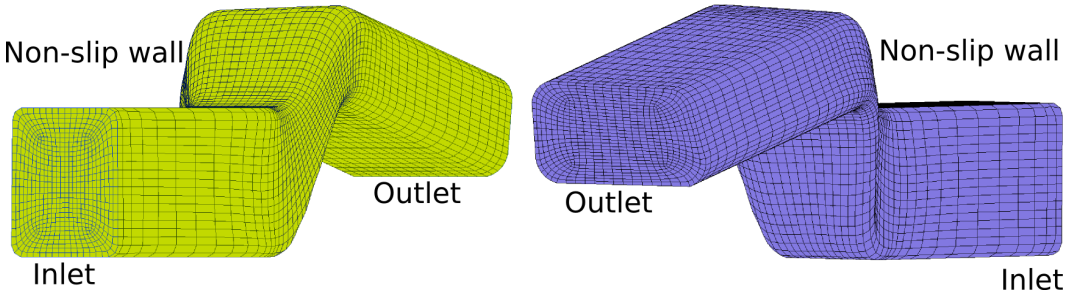


Figure 2.11: S-bend air duct from a VW Golf vehicle

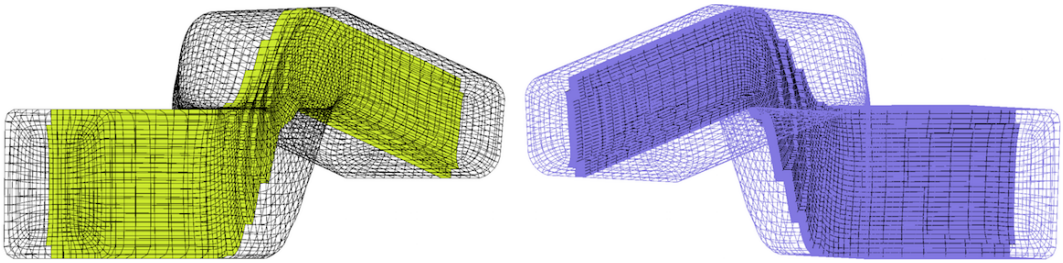


Figure 2.12: S-bend case mesh detail: internal cross section along the duct

Table 2.2: Parameters set for solver comparison

Solver	Algorithm	α_u	α_p	Linear solver
GPDE	SIMPLE	0.9	0.1	BICG/CGSTAB
OpenFOAM	SIMPLE	0.9	0.1	BICG/CGSTAB

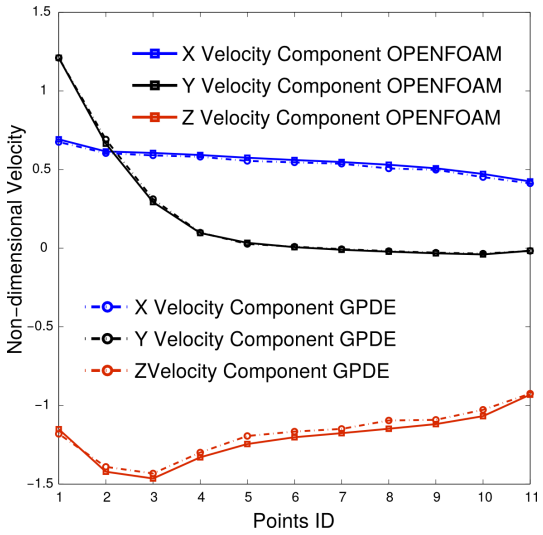


Figure 2.13: Velocity components comparison between GPDE and OpenFOAM solutions

Chapter 3

Discrete adjoint solver

Rather than early stage of shape design, baseline shape optimisation aims to further modify detailed shape changing. Therefore, gradient-based or deterministic numerical method is a good choice to look for local optimum solution with purpose to minimise the objective function. In gradient-based optimisation algorithms, one needs to calculate the gradient $\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\alpha}}$, which is also known as sensitivity with regard to objective function (or cost function) L, and $\boldsymbol{\alpha}$ is control variables (or design variables). In general, flow shape optimisation problem can be stated as

$$Minimise \quad L(\mathbf{W}(\boldsymbol{\alpha}), \boldsymbol{\alpha}) \tag{3.1}$$

constrained by flow state equation:

$$\mathbf{R}(\mathbf{W}(\boldsymbol{\alpha}), \boldsymbol{\alpha}) = 0. \tag{3.2}$$

where $\mathbf{W}$ is the flow state variables, which is also dependent on control variables $\boldsymbol{\alpha}$. For solving the sensitivity (the derivative), methods, including finite difference, tangent linearisation and adjoint method, are widely employed [97].

Finite Difference The simplest way to calculate the gradient is finite difference (FD):

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\alpha}} \approx \frac{L((\mathbf{W} + \delta\mathbf{W}), (\boldsymbol{\alpha} + \delta\boldsymbol{\alpha})) - L(\mathbf{W}, \boldsymbol{\alpha})}{\delta\boldsymbol{\alpha}} \tag{3.3}$$

where $\delta\boldsymbol{\alpha}$ is control variables changing value and $\delta\mathbf{W}$ is perturbed flow state variables. For shape optimisation problems, the number of objective functions is normally much smaller than the number of control variables. Considering m design variables needs to be modified to minimise one scalar cost function, we need to compute m times to

obtain the final gradient vector:

$$\begin{aligned}
\frac{\mathrm{d}L}{\mathrm{d}\alpha_1} &\approx \frac{L((\mathbf{W}+\delta\mathbf{W}_1),(\alpha_1+\delta\alpha_1))}{\delta\alpha_1};\\
\frac{\mathrm{d}L}{\mathrm{d}\alpha_2} &\approx \frac{L((\mathbf{W}+\delta\mathbf{W}_2),(\alpha_1+\delta\alpha_2))}{\delta\alpha_2};\\
&\cdots;\\
\frac{\mathrm{d}L}{\mathrm{d}\alpha_m} &\approx \frac{L((\mathbf{W}+\delta\mathbf{W}_m),(\alpha_m+\delta\alpha_m))}{\delta\alpha_m};
\end{aligned} \tag{3.4}$$

which means that not only the cost function $L_i = L((\mathbf{W}+\delta\mathbf{W}_i),(\alpha_i+\delta\alpha_i))$ needs to be evaluated m times, but the perturbed flow state $(\mathbf{W}+\delta\mathbf{W}_i)$ governed by flow equation $\mathbf{R}((\mathbf{W}+\delta\mathbf{W}_i),(\alpha+\alpha_i))=0$ also needs to be computed m times. And computing m times flow fields can be fairly expensive.

Besides, this straightforward formula Eq. 3.3 shows that the accuracy of gradient depends on the choice of step-width $\delta\boldsymbol{\alpha}$[98]. In order to obtain accurate finite-difference gradient, small step-width $\delta\boldsymbol{\alpha}$ is supposed to be chosen to reduce the truncation error. However, if chosen step-width is excessively small, the dominated round-off error results in meaningless result. The difficulty of choosing the step-width $\delta\boldsymbol{\alpha}$ will be discussed in 3.5.

Tangent Linearisation Sensitivity of the objective function L with respect to $\boldsymbol{\alpha}$ is obtained based on chain rule:

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\alpha}} = \frac{\partial L}{\partial \boldsymbol{\alpha}} + \frac{\partial L}{\partial \mathbf{W}}\frac{\partial \mathbf{W}}{\partial \boldsymbol{\alpha}} \tag{3.5}$$

For ith gradient component,

$$\frac{\mathrm{d}L}{\mathrm{d}\alpha_i} = \frac{\partial L}{\partial \alpha_i} + \frac{\partial L}{\partial \mathbf{W}}\frac{\partial \mathbf{W}}{\partial \alpha_i} \tag{3.6}$$

where $\dfrac{\partial L}{\partial \alpha_i}$ and $\dfrac{\partial L}{\partial \mathbf{W}}$ can be easily obtained via the explicit formula of $L(\mathbf{W},\boldsymbol{\alpha})$, and $\dfrac{\partial \mathbf{W}}{\partial \alpha_i}$ is the perturbation flow state value vector with respect to ith design variable α_i. Here, the flow system offers us additional information of the perturbation field. Taking the derivative of state equation with respective to ith design variable:

$$\frac{\mathrm{d}\mathbf{R}}{\mathrm{d}\alpha_i} = 0 \tag{3.7}$$

which equivalent with

$$\frac{\partial \mathbf{R}}{\partial \mathbf{W}}\frac{\partial \mathbf{W}}{\partial \alpha_i} + \frac{\partial \mathbf{R}}{\partial \alpha_i} = 0 \tag{3.8}$$

If let $-\dfrac{\partial \mathbf{R}}{\partial \alpha_i} = \mathbf{f}_i$ and $\dfrac{\partial \mathbf{W}}{\partial \alpha_i} = \mathbf{u}_i$, the equation above can be expressed as

$$\mathbf{A}\mathbf{u}_i = \mathbf{f}_i. \tag{3.9}$$

where $\mathbf{A} = \dfrac{\partial \mathbf{R}}{\partial \mathbf{W}}$, which is the flow Jacobian. Only perturbing one design variable can bring us the perturbation field vector $\mathbf{u}_i$ and the corresponding right hand side vector $\mathbf{f}_i$, which results in the linear system Eq. (3.9) for solving perturbation field $\dfrac{\partial \mathbf{W}}{\partial \alpha_i}$. Similarly, the cost for computing perturbation field Eq. (3.9) is the same with the primal. Therefore, final cost of sensitivity with respective to all control variables is proportional to the number of control variables.

Adjoint Method As discussed above, the most expensive part for sensitivity computation is times computation of perturbation field $\mathbf{u} = \frac{\partial \mathbf{W}}{\partial \boldsymbol{\alpha}}$. Then it is natural to consider a way how we can obtain final sensitivity without computing perturbation field in to order to save computing resource. Adjoint method offers us the alternative way.

The sensitivity is:

$$\begin{aligned} \frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\alpha}} &= \frac{\partial L}{\partial \boldsymbol{\alpha}} + \frac{\partial L}{\partial \mathbf{W}} \frac{\partial \mathbf{W}}{\partial \boldsymbol{\alpha}} \\ &= \frac{\partial L}{\partial \boldsymbol{\alpha}} + \mathbf{g}^T \mathbf{u} \end{aligned} \tag{3.10}$$

where $\mathbf{g}^T = \frac{\partial L}{\partial \mathbf{W}}$. In order to replace the perturbation matrix $\mathbf{u}$ by given information, one needs to look back on the state-constrain:

$$\mathbf{R}(\mathbf{W}, \boldsymbol{\alpha}) = 0, \tag{3.11}$$

the linearisation with respect to design variable $\boldsymbol{\alpha}$ produces:

$$\frac{\partial \mathbf{R}}{\partial \mathbf{W}} \frac{\partial \mathbf{W}}{\partial \boldsymbol{\alpha}} + \frac{\partial \mathbf{R}}{\partial \boldsymbol{\alpha}} = 0 \tag{3.12}$$

which can be expressed as:

$$\mathbf{A}\mathbf{u} = \mathbf{f} \tag{3.13}$$

where the perturbation is considered as unknown variables $\mathbf{u} = \frac{\partial \mathbf{W}}{\partial \boldsymbol{\alpha}}$, the coefficient matrix is flow Jocobian $\mathbf{A} = \dfrac{\partial \mathbf{R}}{\partial \mathbf{W}}$ and right hand side is $\mathbf{f} = -\dfrac{\partial \mathbf{R}}{\partial \boldsymbol{\alpha}}$. Therefore, $\mathbf{u} = \mathbf{A}^{-1}\mathbf{f}$, which can be introduced back in to Eq. (3.10), and then:

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\alpha}} = \frac{\partial L}{\partial \boldsymbol{\alpha}} + \mathbf{g}^T \mathbf{A}^{-1} \mathbf{f}. \tag{3.14}$$

As already known to us, computing system $\mathbf{Au} = \mathbf{f}$, where $\mathbf{u}$ is a matrix, is expensive. Instead, an alternative equation is solved to get the derivative. If we transpose the last term in the equation above, we get

$$\mathbf{f}^T\mathbf{A}^{-T}\mathbf{g} \tag{3.15}$$

Let $\mathbf{A}^{-T}\mathbf{g}$ equal a new unknown vector $\mathbf{v}$, and there is additional linear equation

$$\mathbf{A}^T\mathbf{v} = \mathbf{g} \tag{3.16}$$

with solution $\mathbf{v} = \mathbf{A}^{-T}\mathbf{g}$ and $\mathbf{g}^T = \mathbf{v}^T\mathbf{A}$. The cost of computing the linear system is similar to the primal system as the coefficient matrix is the transpose of flow Jacobian. Consequently, the final sensitivity is:

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\alpha}} = \frac{\partial L}{\partial \boldsymbol{\alpha}} + \mathbf{v}^T\mathbf{A}\mathbf{A}^{-1}\mathbf{f} \quad = \frac{\partial L}{\partial \boldsymbol{\alpha}} + \mathbf{v}^T\mathbf{f} \tag{3.17}$$

Here the additional linear system $\mathbf{A}^T\mathbf{v} = \mathbf{g}$ is adjoint system and $\mathbf{v}$ is adjoint variable vector. 'Adjoint equivalence' is expressed as:

$$\mathbf{g}^T\mathbf{u} = (\mathbf{A}^T\mathbf{v})^T\mathbf{u} = \mathbf{v}^T\mathbf{A}\mathbf{u} = \mathbf{v}^T\mathbf{f} \tag{3.18}$$

The adjoint method can also be obtained from the viewpoint of the method of Lagrange multipliers, where the adjoint variable is Lagrange multiplier [42][43]. For cost function L, the extrema is at a stationary point, where zero gradient is satisfied:

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\alpha}} = \frac{\partial L}{\partial \boldsymbol{\alpha}} + \frac{\partial L}{\partial \mathbf{W}}\frac{\partial \mathbf{W}}{\partial \boldsymbol{\alpha}} = 0. \tag{3.19}$$

The flow equation is the constraint. Linearising this constraint, one obtains:

$$\frac{\partial \mathbf{R}}{\partial \mathbf{W}}\mathrm{d}\mathbf{W} + \frac{\partial \mathbf{R}}{\partial \boldsymbol{\alpha}}\mathrm{d}\boldsymbol{\alpha} = 0. \tag{3.20}$$

Introducing this constraint back to the cost function gradient with Lagrange multipliers, the differential change in the cost function is:

$$\begin{aligned} \mathrm{d}L &= \frac{\partial L}{\partial \boldsymbol{\alpha}}\mathrm{d}\boldsymbol{\alpha} + \frac{\partial L}{\partial \mathbf{W}}\mathrm{d}\mathbf{W} - \mathbf{v}^T\left(\frac{\partial \mathbf{R}}{\partial \mathbf{W}}\mathrm{d}\mathbf{W} + \frac{\partial \mathbf{R}}{\partial \boldsymbol{\alpha}}\mathrm{d}\boldsymbol{\alpha}\right) \\ &= \left(\frac{\partial L}{\partial \boldsymbol{\alpha}} - \mathbf{v}^T\frac{\partial \mathbf{R}}{\partial \boldsymbol{\alpha}}\right)\mathrm{d}\boldsymbol{\alpha} + \left(\frac{\partial L}{\partial \mathbf{W}} - \mathbf{v}^T\frac{\partial \mathbf{R}}{\partial \mathbf{W}}\right)\mathrm{d}\mathbf{W} \end{aligned} \tag{3.21}$$

where the adjoint system is obtained as:

$$\frac{\partial \mathbf{R}^T}{\partial \mathbf{W}}\mathbf{v} = \frac{\partial L^T}{\partial \mathbf{W}}. \tag{3.22}$$

Eliminating the second term of flow perturbation from Eq. (3.21), it becomes:

$$\mathrm{d}L = \left(\frac{\partial L}{\partial \boldsymbol{\alpha}} - \mathbf{v}^T \frac{\partial \mathbf{R}}{\partial \boldsymbol{\alpha}}\right) \mathrm{d}\boldsymbol{\alpha}. \tag{3.23}$$

which results in the adjoint sensitivity in the same formulation of Eq. (3.17):

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\alpha}} = \frac{\partial L}{\partial \boldsymbol{\alpha}} - \mathbf{v}^T \frac{\partial \mathbf{R}}{\partial \boldsymbol{\alpha}} = \frac{\partial L}{\partial \boldsymbol{\alpha}} + \mathbf{v}^T \mathbf{f}. \tag{3.24}$$

In this chapter, discrete adjoint will be introduced first. Thanks to Automatic Differentiation tool, Tapenade, the discrete adjoint solver is built-up reliably and efficiently for surface sensitivity calculation. In order to improve the discrete adjoint performance, some codes implementation needs to be done for pre-precessing the primal/flow solver. Successively, sensitivity validation among finite-difference, tangent-linear (forward mode) and adjoint (reverse mode) methods verify the discrete adjoint solver.

3.1 Discrete Adjoint Method

For non-adjoint methods, the sensitivity is obtained based on the computation of perturbation field associated with design variables, which means the cost would be considerable when we cope with a big amount of design parameters. However, adjoint methods eschews the computation of the perturbation field and hence the computation cost is independent of the design parameters. Compared with these non-adjoint methods, adjoint methods is popularly accepted with its distinct vantage of efficiency when dissolving massive design parameter cases . From the transonic inviscid three-point optimisation of a Boeing 747 wing [10] to the surface sensitivity calculation of an S-bend air duct [99], the adjoint method has been used in an expanding range of applications. Among many gradient-based optimisation approaches, the adjoint formulation is considered as the most promising one of them whereby the sensitivity of the objective function with respect to an arbitrary number of design variables is obtained with the equivalent of only one additional flow calculation [47].

Finite volume methods typically use semi-discrete formulation that separate time and spatial discretisations, using a variety of fixed-point iterative schemes to reach the steady state.

As discussed above, the sensitivity is expressed as

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\alpha}} = \frac{\partial L}{\partial \boldsymbol{\alpha}} + \mathbf{g}^T \mathbf{u}. \tag{3.25}$$

However this *forward* approach requires M times accumulation of $\mathbf{g}^T\mathbf{u}$ for M design

variables $\boldsymbol{\alpha}$ as we discussed above. With the adjoint variable, the sensitivity is expressed as

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\alpha}} = \frac{\partial L}{\partial \boldsymbol{\alpha}} + \mathbf{v}^T \mathbf{f}. \tag{3.26}$$

where adjoint variable is the solution of equation $\mathbf{A}^T\mathbf{v} = \mathbf{g}$. $\mathbf{A}$ is the flow-Jacobian which is linearised around the fixed-point solution. This formulation is exact, provided the Jacobian $\frac{\partial \mathbf{R}}{\partial \mathbf{W}}$ is differentiated exactly.

The adjoint approach allows to compute this sensitivity more efficiently by transposing and refactoring Eq. (3.26):

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\alpha}}^T = \frac{\partial L}{\partial \boldsymbol{\alpha}}^T + \mathbf{f}^T\mathbf{v} \tag{3.27}$$

The cost of computing this sensitivity only depends on the number of objective functions L, which is typically very small. And the cost of solving the adjoint equation $\mathbf{A}^T\mathbf{v} = \mathbf{g}$ is similar with the primal one. This hence represents a much more economical approach.

In the discrete approach the differentiation of the conservative fluxes residual $\mathbf{R}$ can be achieved using automatic differentiation (AD) tools in reverse mode. In this case the Tapenade AD-tool[100] is used. The SIMPLE algorithm does not compute the full Jacobian, instead the algorithm can be regarded as computing a number of sparse matrix vector products of the type $\frac{\partial \mathbf{R}}{\partial \mathbf{W}}\mathbf{W}$ in a particular block sequence. The adjoint equivalent is achieved by assembling transposed matrix-vector products of the type $\frac{\partial \mathbf{R}}{\partial \mathbf{W}}^T\mathbf{v}$ where $\mathbf{v}$ is the adjoint solution. In order to improve computational efficiency, the fixed-point nature of the iteration allows us to differentiate only the final iteration[101] and linearise the Jacobian around the steady state solution. Furthermore we can eliminate the differentiation invariant elements from the solver stack such as linear solvers and replace them with appropriately modified calls to the same solver[102].

The discrete approach is preferred for sensitivity computation based on two following reasons:

- Based on 'Adjoint equivalence', discrete adjoint method offers adjoint system coefficient matrix (transpose matrix of primal system), which guarantee the similar cost for adjoint calculation and the similar iterative method (fixed-point iteration) for discrete adjoint system.

- According to the chain rule, the final gradient can be assembled by parts of derivatives. Automatic differentiation can be applied for linearised discrete CFD system in parallel, which avoids breaking the CFD codes, deriving extra adjoint equation and re-discretizing.

- Once auto differentiation as been set up (which is a significant effort), then new models implemented in the same way as the existing code are automatically differentiated. It is very effective for code maintenance.

These issues are discussed in detail in next two subsections.

3.1.1 General outputs of sensitivity propagation

As we can see from adjoint method derived in last section, its advantage is avoiding computing the most expensive part $\frac{\partial \mathbf{W}}{\partial \boldsymbol{\alpha}}$ which is flow vector derivative with respect to design variables vector. The adjoint variable is an intermediate variable indeed, and look back on Eq. (3.18) gives:

$$\left(\frac{\partial L}{\partial \mathbf{W}}\right)\frac{\partial W}{\partial \boldsymbol{\alpha}} = \mathbf{v}^T\mathbf{f} \tag{3.28}$$

For simplification purpose, objective derivative with respect to flow is defined as

$$\left(\frac{\partial L}{\partial \mathbf{W}}\right)^T = \overline{\mathbf{W}}$$

and the flow variable derivative with respect to design variable is defined as

$$\frac{d\mathbf{W}}{d\boldsymbol{\alpha}} = \dot{\mathbf{W}}.$$

If scalar objective function is defined as $L = L_\alpha + L_w$, $\frac{dL_w}{d\mathbf{W}}$ is expensive term that contains discrete adjoint component $\overline{\mathbf{W}} = \left(\frac{dL_w}{d\mathbf{W}}\right)^T$. Thus, this gradient process can be viewed as:

$$\boldsymbol{\alpha} \rightarrow \mathbf{W} \rightarrow L_w.$$

Based on the chain rule, the gradient of composite function L_w is:

$$\frac{\mathrm{d}L_w}{\mathrm{d}\boldsymbol{\alpha}} = \frac{\mathrm{d}L_w}{\mathrm{d}\mathbf{W}}\frac{\mathrm{d}\mathbf{W}}{\mathrm{d}\boldsymbol{\alpha}} \tag{3.29}$$

And the this part sensitivity can be expressed as:

$$\dot{L_w}\overline{L_w} = \overline{\mathbf{W}}^T\dot{\mathbf{W}} \tag{3.30}$$

For any other intermediate variable $X(\boldsymbol{\alpha})$ dependent on design variable $\boldsymbol{\alpha}$, more general gradient formula can be expressed:

$$\frac{\mathrm{d}L_w}{\mathrm{d}\boldsymbol{\alpha}} = \frac{\mathrm{d}L_w}{\mathrm{d}\mathbf{W}}\frac{\mathrm{d}\mathbf{W}}{\mathrm{d}X}\frac{\mathrm{d}X}{\mathrm{d}\boldsymbol{\alpha}} \tag{3.31}$$

which leads to linear propagation:

$$\dot{L_w} = \frac{\mathrm{d}L_w}{\mathrm{d}\mathbf{W}}\frac{\mathrm{d}\mathbf{W}}{\mathrm{d}X}\frac{\mathrm{d}X}{\mathrm{d}\boldsymbol{\alpha}}\dot{\boldsymbol{\alpha}} \tag{3.32}$$

According to the definition:

$$\dot{X} = \frac{\mathrm{d}X}{\mathrm{d}\boldsymbol{\alpha}}\dot{\boldsymbol{\alpha}}, \quad \dot{\mathbf{W}} = \frac{\mathrm{d}\mathbf{W}}{\mathrm{d}X}\dot{X}, \quad \dot{L_w} = \frac{\mathrm{d}L_w}{\mathrm{d}\mathbf{W}}\dot{\mathbf{W}},$$

and AD (Automatic Differentiation) working in *forward* process can be conducted:

$$\dot{\boldsymbol{\alpha}} \to \dot{X} \to \dot{\mathbf{W}} \to \dot{L_w}.$$

Meanwhile, according to definition of $\overline{\boldsymbol{\alpha}}$, $\overline{X}$and $\overline{\mathbf{W}}$:

$$\begin{aligned} \dot{L_w} &= \frac{\mathrm{d}L_w}{\mathrm{d}\mathbf{W}}\dot{\mathbf{W}} = \overline{\mathbf{W}}^T\dot{\mathbf{W}} \\ &= \frac{\mathrm{d}L_w}{\mathrm{d}X}\dot{X} = \overline{X}^T\dot{X} \\ &= \frac{\mathrm{d}L_w}{\mathrm{d}\boldsymbol{\alpha}}\dot{\boldsymbol{\alpha}} = \overline{\boldsymbol{\alpha}}^T\dot{\boldsymbol{\alpha}} \end{aligned} \tag{3.33}$$

where $\dot{\boldsymbol{\alpha}} = 1$ and its adjoint counterparts $\overline{\boldsymbol{\alpha}}$ equals the transpose of final gradient. Once the 'chain relationship', like $\frac{\mathrm{d}X}{\mathrm{d}\boldsymbol{\alpha}}$, $\frac{\mathrm{d}\mathbf{W}}{\mathrm{d}X}$ and $\frac{\mathrm{d}L_w}{\mathrm{d}\mathbf{W}}$ are fixed (determined by primal calculation), the gradient can be obtain via AD working in *reverse* process:

$$\begin{aligned} \overline{\boldsymbol{\alpha}} &= \frac{\mathrm{d}L_w}{\mathrm{d}\boldsymbol{\alpha}}^T \\ &= \left(\frac{\mathrm{d}L_w}{\mathrm{d}\mathbf{W}}\frac{\mathrm{d}\mathbf{W}}{\mathrm{d}X}\frac{\mathrm{d}X}{\mathrm{d}\boldsymbol{\alpha}}\right)^T \\ &= \left(\frac{\mathrm{d}X}{\mathrm{d}\boldsymbol{\alpha}}\right)^T\left(\frac{\mathrm{d}\mathbf{W}}{\mathrm{d}X}\right)^T\left(\frac{\mathrm{d}L_w}{\mathrm{d}\mathbf{W}}\right)^T. \end{aligned} \tag{3.34}$$

According to the definition $\overline{L_w} = 1$ and:

$$\overline{\mathbf{W}} = \left(\frac{\mathrm{d}L_w}{\mathrm{d}\mathbf{W}}\right)^T\overline{L_w}, \quad \overline{X} = \left(\frac{\mathrm{d}\mathbf{W}}{\mathrm{d}X}\right)^T\overline{\mathbf{W}}, \quad \overline{\boldsymbol{\alpha}} = \left(\frac{\mathrm{d}X}{\mathrm{d}\boldsymbol{\alpha}}\right)^T\overline{X}$$

Thus, the adjoint quantities in the AD community can be propagated in *reverse* or

backward accumulation:

$$\overline{L_w} \rightarrow \overline{\mathbf{W}} \rightarrow \overline{X} \rightarrow \overline{\boldsymbol{\alpha}}$$

Consequently, the general sensitivity propagation gives Automatic Differentiation chance to build up computational graph, which ease the final gradient calculation via assembling parts of differentiation automatically. In general, assuming arbitrary quantity differentiation in *forward* mode expressed:

$$\dot{X} = \mathbf{A}\dot{\mathbf{W}}, \tag{3.35}$$

the adjoint sensitivities satisfy:

$$\overline{\mathbf{W}} = \mathbf{A}^T\overline{X} \tag{3.36}$$

which is accumulated in *reverse* mode; whereas the output derivatives becomes its adjoint sensitivity inputs in *reverse* mode formulation.

3.1.2 Fixed point iteration

In CFD gradient-based optimisation problem, the intermediate flow variable $\mathbf{W}$of objective gradient is the solution of flow equation $\mathbf{R}(\mathbf{W}, \boldsymbol{\alpha}) = 0.$, that means that $\mathbf{W}$ can not be evaluated by successive explicit operations. Have a close look at the objective derivative with respect to intermediate flow state $\mathbf{W}$, which requires fixed-point iterative algorithm to achieve.

Normally $\mathbf{R}(\mathbf{W}, \boldsymbol{\alpha}) = 0$ is computed via:

$$\mathbf{W}^{n+1} = \mathbf{W}^n - \mathbf{P}(\mathbf{W}^n, \boldsymbol{\alpha})\mathbf{R}(\mathbf{W}^n, \boldsymbol{\alpha}) \tag{3.37}$$

where $\mathbf{P}$ is preconditioner matrix, like the one applied in SIMPLE algorithm as discussed in Chapter 4. Based on Taylor expansion of flow equation at final iteration $\mathbf{W}^{n+1}$ gives:

$$\mathbf{R}(\mathbf{W}^{n+1}) \approx \mathbf{R}(\mathbf{W}^n) + \frac{\partial \mathbf{R}}{\partial \mathbf{W}^n}\delta\mathbf{W} \tag{3.38}$$

and then

$$\delta\mathbf{W} = \mathbf{W}^{n+1} - \mathbf{W}^n \approx \left(\frac{\partial \mathbf{R}}{\partial \mathbf{W}^n}\right)^{-1}\left[\mathbf{R}(\mathbf{W}^{n+1}) - \mathbf{R}(\mathbf{W}^n)\right]. \tag{3.39}$$

Because of fully convergence, final residual $\mathbf{R}(\mathbf{W}^{n+1}) = 0$ is introduced back into the equation above:

$$\mathbf{W}^{n+1} \approx \mathbf{W}^n - \left(\frac{\partial \mathbf{R}}{\partial \mathbf{W}^n}\right)^{-1}\mathbf{R}(\mathbf{W}^n) \tag{3.40}$$

which is compared with Eq. (3.37), and we can see that $\mathbf{P}$ is normally the approximation of inverse of flow Jocobian $\dfrac{\partial \mathbf{R}}{\partial \mathbf{W}}$.

Differentiating the flow equation $\mathbf{R}(\mathbf{W}, \boldsymbol{\alpha}) = 0$ leads to:

$$\frac{\partial \mathbf{R}}{\partial \mathbf{W}} \frac{\partial \mathbf{W}}{\partial \boldsymbol{\alpha}} + \frac{\partial \mathbf{R}}{\partial \boldsymbol{\alpha}} = 0 \tag{3.41}$$

which also can be expressed as:

$$\mathbf{A}\dot{\mathbf{W}} + \dot{\mathbf{R}} = 0. \tag{3.42}$$

In order to solve the equation above, the iterative method can be expressed as:

$$\dot{\mathbf{W}^{n+1}} = \dot{\mathbf{W}^{n}} - \tilde{\mathbf{P}} \left(\mathbf{A}\dot{\mathbf{W}^{n}} + \dot{\mathbf{R}^{n}} \right) \tag{3.43}$$

When computation converges, one achieves $\tilde{\mathbf{P}} = \mathbf{P} = \mathbf{A}^{-1}$. The converged solution:

$$\begin{aligned} \dot{\mathbf{W}} &= \dot{\mathbf{W}^{*}} - \mathbf{A}^{-1} \left(\mathbf{A}\dot{\mathbf{W}^{*}} + \dot{\mathbf{R}^{*}} \right) \\ &= \dot{\mathbf{W}^{*}} - \mathbf{A}^{-1}\mathbf{A}\dot{\mathbf{W}^{*}} - \mathbf{A}^{-1}\dot{\mathbf{R}^{*}} \\ &= \dot{\mathbf{W}^{*}} - \dot{\mathbf{W}^{*}} - \mathbf{A}^{-1}\dot{\mathbf{R}^{*}} \\ &= -\mathbf{A}^{-1}\dot{\mathbf{R}^{*}} \end{aligned} \tag{3.44}$$

that is equivalent with:

$$\mathbf{A}\dot{\mathbf{W}} + \dot{\mathbf{R}} = 0 \tag{3.45}$$

and the derivative linear system is solved via the similar iterative steps. Thank to the advantage of similar iterative matrix applied in non-linear flow equation and linear sensitivity equation, AD codes can be implemented by applying the same updating driver for both equations.

Meanwhile, because of the general relationship between input/output of *forward* and *reverse* quantities, the adjoint equation can be derive from Eq. (3.42) accordingly as:

$$\mathbf{A}^{T}\overline{\mathbf{R}} + \overline{\mathbf{W}} = 0 \tag{3.46}$$

since whose solution is:

$$\overline{\mathbf{R}} = -\left(\mathbf{A}^{-1}\right)^{T} \overline{\mathbf{W}} \tag{3.47}$$

Similarly, in order to solve Eq. (3.46), the same iterative frame work can be applied for adjoint equation as :

$$\overline{\mathbf{R}^{n+1}} = \overline{\mathbf{R}^{n}} - \tilde{\mathbf{P}}^{T} \left(\mathbf{A}^{T}\overline{\mathbf{R}^{n}} + \overline{\mathbf{W}^{n}} \right) \tag{3.48}$$

whose iteration has the same rate of convergence as the liner system since the system matrices $\mathbf{I} - \mathbf{PA}$ and $I - \mathbf{P}^T\mathbf{A}^T$ have the same eigenvalue. $\mathbf{R}$ is the adjoint variable, which equals $-\mathbf{v}$ as discussed before. This shows the improvement on flow computation has positive impacts on discrete adjoint intermediately as well. The stabilisation methods applying on non-linear equation convergence stabilise sensitivity convergence, too.

3.2 AD tool: Tapenade

The construction of discrete adjoint solver is derived theoretically above. However, the manual implementation of each intermediate variables is tedious and error-prone [103]. AD tool is applied to generate differentiation codes in an automatic approach rather than hand-differentiated one, which can be used as "black-box" tool, by fed with primal codes and "interpreting" them to differentiated ones in *forward* or *reverse* mode. In our case, Tapenade is called to generate differentiated codes for constructing discrete adjoint solver. Tapenade is an Automatic Differentiation Engine developed at INRIA Sophia-Antipolis by the Tropics then Ecuador teams[104]. This strategy of differentiated program interpretation is *source code transformation*. In this way, both of the primal and adjoint codes can be complied in the similar way, and the accessible differentiated source codes are opened for developer further modifying and debugging conveniently. Our in-house CFD codes written in Fortran95, which can be recognised by Tapenade. Even thought AD tool offers an easy way to calculate the gradient, the accurate calculation and efficient performance of discrete adjoint solver is determined by well-understanding the primal/flow algorithm and solver implementation and well-manipulating AD tool. For CFD primal codes, complex structure and a certain scale of codes amount requires specific command-line options and setting programs in primal codes for AD tool to "interpret" the original codes sophisticatedly.

3.3 Adjoint code implementation

Before differentiating primal codes, some preparation need to be done. First of all, the *top differentiation routine* needs to be specified for AD before feeding the primal. The *top differentiation routine* in GPDE is the top-level procedure of SIMPLE algorithms: outer-iteration loop. As we discussed above, this iterative stepping for primal, tangent linear and discrete adjoint solvers is the same, so the driver routine for differentiated codes can be modified manually based on the primal codes for more efficient solver performance. Secondly, the *dependent output* variables

and *independent input* variables should be set for AD to identify objective function and control variables.

3.3.1 Adjoining independent iterative loops

For steady incompressible flow computed by SIMPLE-family algorithms, the outer iteration updates velocity and pressure fields to the converged unchanged solution. This linearised non-linear system is dependent loop, because the solution is depends on the previous results. Take convection term in momentum for example. This non-linear term $\rho\mathbf{uu}$ linearised numerically by $\rho\mathbf{u}*\mathbf{u}$ where $\mathbf{u}*$ is the obtained from last outer iteration. If Tapenade differentiates the primal codes in "brute-force" way, all variables at each iteration (ϕ^1,ϕ^2,..., ϕ^m) will be pushed into stack. However, steady flow computation is the fixed point iteration. As we discussed in Sec. 3.1.2, discrete adjoint system has the same system matrix with the converged one in primal system, so only the last iteration solution is required to be stored for adjoint solver. In this way, memory requirement reduces significantly.

Besides of this dependent loop, there are a big amount of independent iteration loops in primal solver. The independent iterative loop is the loop that can be parallelized. In order to make every effort to reduce the memory, lots of independent iterations needs to be pointed out to Tapenade, because independent iteration loops do not need to tape/store each step's values. With purpose of saving memory and efficient performance, **!$AD II-LOOP** pragma/directive is coded intermediately before each independent iteration loops in order to "tell" Tapnade that the following loop has variables that do not depend on those from another iteration.

3.3.2 Linear Solver Implementation

Because all linear systems can be expressed as $\mathbf{Ax} = \mathbf{b}$, linear solvers are normally implemented independently as a flexibly called package in certain scale codes. AD could differentiate through the linear solver, Which (in general) will give the correct derivative, but the differentiated code of this part is sophisticated and can not work efficiently. An alternative way is to use modification codes to replace the linear solver codes differentiated by AD tool. Manual modification for tangent linear and discrete adjoint solver is necessary and simple to implement. For tangent linear method, based on total derivative:

$$\mathrm{d}\mathbf{Ax} + \mathbf{A}\mathrm{d}x = \mathrm{d}\mathbf{b} \tag{3.49}$$

which can be expressed as:

$$\dot{\mathbf{A}}\mathbf{Ax} + \mathbf{A}\dot{\mathbf{x}} = \dot{\mathbf{b}} \tag{3.50}$$

where the perturbed value $\dot{\mathbf{x}}$ or dx can be obtained:

$$\dot{\mathbf{x}} = \mathbf{A}^{-1}(\dot{\mathbf{b}} - \dot{\mathbf{A}}x) \tag{3.51}$$

and then the tangent linear code can be modified as:

```
subroutine Tangent_linear_solver (A, dA, x, dx, b, db, n)
      call Linear_solver (A, x, b, n)
      do i=1, n
           temp(i)=db(i)
           do j=1, n
                temp(i)=temp(i)-dA(i, j)x(j)
           end do
           call Linear_solver(A, dx, temp, n)
      end do
end subroutine
```

Normally the system matrix $\mathbf{A}$ and right hand side $\mathbf{b}$ dependent on control variable are calculated explicit which means that perturbed terms d$\mathbf{A}$ and d$\mathbf{b}$ are obtained straightforward rather than dx. According to the implementation mentioned above, dx can be computed via modified tangent linear solver subroutine in order to avoid calculating inverse matrix $\mathbf{A}^{-1}$.

For discrete adjoint method, $\overline{\mathbf{x}}$, $\overline{\mathbf{b}}$ and $\overline{\mathbf{A}}$ stand for their adjoint variables. Based on general output, adjoint linear system to determine $\mathbf{b}$ can be obtained straightforwardly as:

$$\mathbf{A}^T\overline{\mathbf{b}} = \overline{\mathbf{x}}. \tag{3.52}$$

In primal system, $\mathbf{x}$ is determined by $\mathbf{A}$ and $\mathbf{b}$, so accordingly, in adjoint (or *reverse* mode) system, $\overline{\mathbf{A}}$ and $\overline{\mathbf{b}}$ are determined by $\overline{\mathbf{x}}$. It needs to be noticed that we are not talking about fixed point iteration since the system matrix $\mathbf{A}$ here is intermediate variable and its adjoint part also needs to be accumulated for final gradient solution. With convenience purpose, $\circ$ is noted as element-wise dot product. In AD community,

$$\dot{\mathbf{x}} \circ \overline{\mathbf{x}} = \dot{\mathbf{A}} \circ \overline{\mathbf{A}} + \dot{\mathbf{b}} \circ \overline{\mathbf{b}} \tag{3.53}$$

introducing $\dot{\mathbf{x}} = \mathbf{A}^{-1}(\dot{\mathbf{b}} - \dot{\mathbf{A}}\mathbf{x})$ into equation above:

$$\begin{aligned} \dot{\mathbf{A}} \circ \overline{\mathbf{A}} + \dot{\mathbf{b}} \circ \overline{\mathbf{b}} &= (\mathbf{A}^{-1}(\dot{\mathbf{b}} - \dot{\mathbf{A}}\mathbf{x})) \circ \overline{\mathbf{x}} \\ &= (\dot{\mathbf{b}} - \dot{\mathbf{A}}\mathbf{x}) \circ ((\mathbf{A}^{-1})^T\overline{\mathbf{x}}) \\ &= \dot{\mathbf{b}} \circ (\mathbf{A}^{-1})^T\overline{\mathbf{x}} - \dot{\mathbf{A}}\mathbf{x} \circ (\mathbf{A}^{-1})^T\overline{\mathbf{x}} \end{aligned} \tag{3.54}$$

and then we can obtain

$$\dot{\mathbf{A}} \circ \overline{\mathbf{A}} = -\dot{\mathbf{A}}\mathbf{x} \circ (\mathbf{A}^{-1})^T \overline{\mathbf{x}} \tag{3.55}$$

and

$$\dot{\mathbf{b}} \circ \overline{\mathbf{b}} = \dot{\mathbf{b}} \circ (\mathbf{A}^{-1})^T \overline{\mathbf{x}} \tag{3.56}$$

which implies $\mathbf{A}^T \overline{\mathbf{b}} = \overline{\mathbf{x}}$ equivalent with the general outputs' result mentioned before. Furthermore, the former equation brings

$$\dot{\mathbf{A}} \circ \overline{\mathbf{A}} = -\dot{\mathbf{A}}\mathbf{x} \circ \overline{\mathbf{b}} \tag{3.57}$$

where indicates that element $A_{i,j}$ in matrix $\overline{\mathbf{A}}$ can be calculated via

$$\overline{A}_{i,j} = -x_j \overline{\mathbf{b}}_i. \tag{3.58}$$

Therefore, the implementation of adjoint linear system can be manually coded as:

```
subroutine Adjoint_linear_solver (A, Ab, x, xb, b, bb, n)
      call Matrix_transpose (A, AT)
      call Linear_solver (AT, temp, xb, n)
      do i=1, n
         bb(i)=bb(i)+temp(i)
      end do
      call Linear_solver (A, x, b, n)
      do i=1, n
         do j=1, n
         Ab(i, j)=Ab(i, j)-x(j)temp(j)
         end do
         xb(i)=0.0
      end do
end subroutine
```

which implementation solves both the primal and adjoint linear systems and where both of system matrix and right hand side are active. And linear equation is assumed to be solved exactly.

3.4 Discrete Adjoint Solver in GPDE

Applying AD tool as a "black-box" to primal codes without any further manual treatment to differentiated codes is the simplest "brute-force" approach. Fixed point iteration leads the discrete adjoint solver built via Tapenade AD tool to be implemented in a more compact way and to perform more efficiently compared with

"brute-force" adjoint solver. According to FPI theory, the converged primal solution do not change any more and for discrete adjoint system only last outer iteration solution (like, velocity and its gradient values, pressure and its gradient values and pressure correction and gradient values) is needed. Tapenade applying source code transformation strategy to generate adjoint codes, which push necessary variables to the stack; and when it comes to adjoint running, all stacked values are popped out at the first adjoint outer iterations. For SIMPLE-family algorithms, the primal loop is

```
Do i = 1, n
 call momentum_equation
 call continuity_equation
 if (last iteration) pushreal8array(U,p,...,p')
End do
call objective
```

And then the differentiated adjoint loop in *reverse* mode gives

```
call objective_b
Do i = 1, n
  if (i==1) call popreal8array(U,p,...,p')
  call continuity_equation_b
  call momentum_equation_b
End do
```

Theoretically, once the primal converges, pressure correction value converges to zero. The outer iteration (FPI) can be modified according to FPI via post-pressing differentiated codes, but normally in segregated algorithms' two major linear systems are differentiated by Tapenade respectively in a "brute-force" way, which means that the reversed continuity subroutine needs to solve two linear systems: pressure correction equation of primal part and pressure equation adjoint part.

```
SUBROUTINE CONTINUITY_EQUATION_B
 CALL PRESSURE_CORRECTION_SETUP
 CALL LIN_EQUATION_SOLVER
 PUSH(p'..)

 CALL PRESSURE_CORRECTION_SETUP_B
 POP(p'..)
 CALL LIN_EQUATION_SOLBER_B
END SUBROUTINE
```

that is because Tapenade works in the way which copies primal codes first and generates differentiated codes. And even inside the differentiated codes' part, Tapenade tries its best to keep the primal converged values from being overwritten. However, it results in redundant codes and further weakens adjoint performance as well. When solving discrete adjoint system, pressure correction could be set to zero and it is unnecessary to be involved in adjoint part. Meanwhile, the pressure correction values is saved in the stack before calling the reversed continuity subroutine, so recomputing is unnecessary as well. Besides, the most expensive component in SIMPLE-family algorithms is to compute pressure-correction Poisson's equation. If we cancel out the primal part of pressure correction linear system in reversed continuity, the pruned adjoint solution converges to the same with the full one theoretically and the CPU time will decrease. Therefore, the discrete adjoint part can be further improved. Sensitivity results from full and pruned "brute-force" discrete adjoint solver are compared as shown in Tab. 3.1. The adjoint linear solver subroutine with '_B' suffix needs to be replaced by the manual modified one, which has been introduced in 3.3.2.

Table 3.1: Comparison between full and pruned "brute-force" discrete adjoint sensitivity

Method	Full	Pruned	$Error_{abs}$
Point x	-1.5512339877742789E-004	-1.5512339878992116E-004	1.249326163355047E-014
Point y	9.6122716582783122E-004	9.6122716588574355E-004	-5.791233378266147E-014
Point z	3.3823810162038798E-004	3.3823810165102872E-004	-3.064074601683009E-014

3.5 Sensitivity validation

Global residual is defined as the maximum value between momentum residual R_m and pressure correction residual$R_{p'}$:

$$R_{global} = max(R_m, R_{p'}) \tag{3.59}$$

And the criteria of convergence is determined by absolute global residual rather than history relative one and set as 10^{-10}. Inner linear systems, momentum and continuity equations, apply the relative residual as tolerance:

$$Tolerance_{inner} = \frac{R^0_{in}}{R^n_{in}} \tag{3.60}$$

and the final inner residual R^n_{in} is saved as counterpart global one (R^m_m or $R^m_{p'}$), where subscript m is the outer iteration number. Applying relative residual reduction has the advantage of avoiding considering the difference order between the primal and

adjoint residual. S-bend case is carried out on grid with the scale of 47k cells, and Re=60. Cost function is inlet-outlet mass averaged total pressure loss:

$$L = \frac{\int_{inlet} \rho u p_{inlet}^{total} dS}{\int_{inlet} \rho u dS} - \frac{\int_{outlet} \rho u p_{outlet}^{total} dS}{\int_{outlet} \rho u dS} \tag{3.61}$$

The mesh coordinates are control variables. For gradient validation, results from FD, tangent linear (which is forward mode of automatic differentiation) and adjoint (which is reverse mode of automatic differentiation) are compared. The gradient calculated via FD method is:

$$\frac{\mathrm{d}f}{\mathrm{d}x} \approx \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{3.62}$$

where the perturbed scalar $x + \delta x$ is one geometry coordinate component picked up at one mesh point.

Table 3.2: FD results based on different step-widths

Δx	Gradient
10^{-3}	-6.1025183129004290E-005
10^{-4}	-6.1034537424120558E-005
10^{-5}	-6.1019989061605816E-005
10^{-6}	-6.1007199292361599E-005
10^{-7}	-6.0955684944013496E-005

For the same geometric component, tangent linear and adjoint solutions are shown in Tab. 3.3. where the FD result is based on step-width of 10^{-4} in Tab. 3.2, and

Table 3.3: Sensitivity validation among FD, tangent linear and discrete adjoint

Method	FD	Tangent	Adjoint (10^{-10})
Sensitivity	-6.1034537424120558E-005	-6.1034951843335452E-005	-6.1034952501251889E-005
$Error_{abs}$	4.144192148938775E-10	0.0	6.579164364456554E-13

absolute error is obtained with the tangent linear result as reference. As mentioned above, FD method is picky for step-width, and for difference case, choosing step-width is uncertain.

Adjoint result in Tab. 3.4 is obtained via applying 10^{-10} and 10^{-6} as tolerance for inner iteration after same outer iteration loops. The absolute error is difference from fully converged tangent sensitivity. From this comparison we can safely achieve the conclusion that in adjoint loop, inner systems need to be converged exactly although ones in primal loop only need to 2-3 magnitudes. However, this conclusion is based on the standard implementation of adjoint linear system in 3.3.2, which is assumed that linear equation is solved exactly. In SIMPLE loop, linear systems are not computed

Table 3.4: Adjoint sensitivity comparison between different tolerance for inner linear system

Setting	Adjoint (10^{-10})	Adjoint (10^{-6})
Sensitivity	-6.1034952501251889E-005	-6.1034297404640722E-005
$Error_{abs}$	6.579164364456554E-13	6.544386947304423E-10

to quite deep residual at each outer iteration, so the suggested manual coded adjoint linear solver subroutine can be further modified. And adjoint counterpart xb of unknown variable x is unnecessary to reset as zero. Base on this modification of standard implementation, inner tolerance in adjoint loop of SIMPLE algorithm is not required to be set as machine precision.

Let the system converge to difference tolerance. Tangent sensitivity obtained at 417 step outer iterations has small difference from sensitivity obtained from 3000 outer iterations, seen in Tab. 3.5. From convergence curve of this test case shown in

Table 3.5: Tangent sensitivity comparison between different outer-loop iterations

Setting	Tangent (417 Outer Iterations)	Tangent (3000 Outer Iterations)
Sensitivity	-6.1034951843335452E-005	-6.1034952524532894E-005
$Error_{abs}$	0.0	6.811974419328008E-13

Fig. 3.1, at 417 outer iteration, system does not converge to exact fixed-point solution rather than it at 3000 out iteration where convergence curve becomes flat. However, from the comparison of sensitivity, two flow fields do not show big difference. For some industrial cases, the primal solution and also the sensitivity solution is not required to converge to machine precision because user-defined tolerance, like 10^{-7} in the test case, the solution is accurate enough and acceptable.

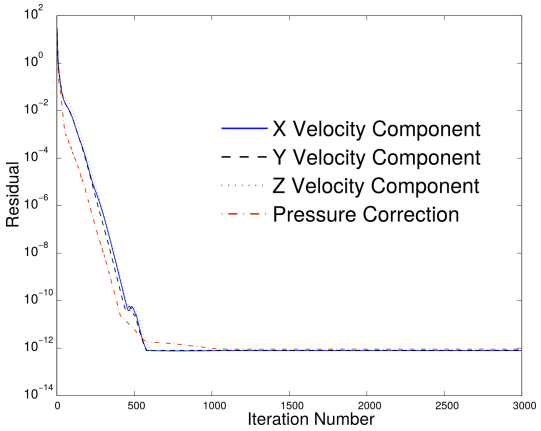


Figure 3.1: Convergence of the solution process for the laminar flow through a s-bend channel on a grid with 47k CV

Chapter 4

Stabilisation of discrete adjoint

For flow solver development, robust and stable performance is the major objective. The flow solver is developed in order to work on a wide range of applications, and for cases that are difficult to converge, the flow solver should have many features to stablise the computation. SIMPLE algorithm often exhibits poor stability, which leads to diverging or only converging to limit-cycle oscillations (LCO) in a lot of industrial cases[105]. There can be several reasons which cause the numerical or physical instability in a system. Taking mesh for example, low quality of mesh due to either relatively coarse or excessively skewed crossing large flow variations may result in numerical error and divergence of the linearised system. Furthermore, although mesh is fine enough to resolve certain local flow unsteadiness in a particular area, it might still let steady-state computation fail to fully converge. Separation bubbles, for instance, have only very loose physical coupling with the bulk ow, but the slightly unsteady flow causes steady solver diverges or only converges to LCO[106]. Before fully converged state or during the LCO, the discrete adjoint solver is likely to be not contractive but to exhibit a number of eigenvalues with magnitude larger than one as a consequence, and the interactive scheme for the adjoint will fail once the error modes accociated with those eigen values have grown sufficiently[107].

In order to emhance the robustness and stability of imcompressible flow solver, a family of segregated algorithms derived from SIMPLE algorithm have been explored[108, 109]. Among them, SIMPLEC(SIMPLE-Consistent)[33] and PISO[110] are the most popular algorithms and milestone progress in practical engineering computations. These approaches are quite similar in that the momentum equation is solved for each velocity component separately with an estimated pressure field. The resulting divergence error then produces a source term in the pressure-correction equation, which is used to update the pressure. These methods are differentiated via how the pressure correction is coupled back to the velocity field[111]. Additonally, these pressure-correction methods are also members of family of pressure-Schur complement (PSC) methods. In SIMPLE algorithm, PSC in the pressure-correction equation arises from

the elimination of the off-diagonal block in continuity equation, where the neighbouring cells' influence is neglected, seen in Eq. 2.19; while SIMPLEC and PISO use better preconditioned PSC [112, 113] by approximating the effects from the neighbouring cells. In addition to the segregated approaches, a fully coupled formulation[114–116] has also been explored to improve the flow solver but at a high computational cost.

The discretion scheme also affects the convergence of the solver and it varies with respect to different linearised systems. The poor accuracy on distorted meshes of the typical cell-centred finite volume discretisations for the second-order pressure-correction equation leads to bad convergence since the numerical error for fluxes accumulation damages the conservation. The typically chosen face-based implementation is only second-order accurate on regular and orthogonal meshes, and cases on skew grids fail to converge due to the inaccuracy of pressure gradient[117]. An alternative is to use the duality-exact schemes which interpolate or recompute values on the dual grid[118]. These schemes allow a linearly-exact gradient computation, which significantly increases the storage cost of the numerical scheme.

In this chapter we explore the strategies from different aspects to stabilize the discrete adjoint by enhancing stabilization of the incompressible primal solver. Firstly the bridge between SIMPLE-like algorithms and PSC methods is built to offer an alternative point of view for pressure-based flow solver development. Moreover, skewness correction methods are introduced to improve the affordability of these flow solvers applied on a wide range of mesh quality. Finally, within these SIMPLE-like algorithms, to smooth the flow field and improve the global convergence, advanced linear solvers are investigated and implemented.

4.1 SIMPLE-like algorithm from view of pressure Schur complement

In Chapter 2, discrete SIMPLE algorithm has been derived from the angle of "prediction-correction" idea, whereas from another mathematical point of view an alternative analysis approach is introduced. Incompressible steady-state Navier-Stokes equations can be expressed by matrix description, and this velocity-pressure coupling system $\mathbf{AW} = \mathbf{b}$ is:

$$\underbrace{\begin{bmatrix} \mathbf{F} & \mathbf{G} \\ \mathbf{D} & 0 \end{bmatrix}}_{\mathbf{A}} \cdot \underbrace{\begin{bmatrix} \mathbf{u} \\ p \end{bmatrix}}_{\mathbf{W}} = \underbrace{\begin{bmatrix} \mathbf{f} \\ 0 \end{bmatrix}}_{\mathbf{b}} \tag{4.1}$$

where $\mathbf{F}$ is the coeffiecient matrix containing both convective and diffusive components in velocity equation; $\mathbf{G}$ the gradient operator and $\mathbf{D} = \mathbf{G}^T$ is the divergence operator; $\mathbf{u}$ and p are respectively velocity vector and pressure; $\mathbf{f}$ is external forces

or sources term. The first convective term is quadratic non-linear, which can be linearised via so-called Newton correction and Picard's method[119].

Prediction-correction strategy leads to:

$$\mathbf{u} = \mathbf{u}^* + \mathbf{u}', \ p = p^* + p' \tag{4.2}$$

where $\mathbf{u}^*$ is intermediate velocity that satisfies momentum equation with initial guess pressure:

$$\mathbf{F}\mathbf{u}^* + \mathbf{G}p^* = \mathbf{f} \tag{4.3}$$

and then $\mathbf{u}^* = \mathbf{F}^{-1}(\mathbf{f} - \mathbf{G}p^*)$. The relationship between $\mathbf{u}'$ and p' is built up based on subtraction of

$$\begin{aligned} \mathbf{F}\mathbf{u} + \mathbf{G}p &= \mathbf{f} \\ \mathbf{F}\mathbf{u}^* + \mathbf{G}p^* &= \mathbf{f} \end{aligned} \tag{4.4}$$

and we obtain

$$\begin{aligned} \mathbf{F}\mathbf{u}' + \mathbf{G}p' &= 0 \\ \mathbf{u}' &= -\mathbf{F}^{-1}\mathbf{G}p' \end{aligned} \tag{4.5}$$

Applying divergence operator $\mathbf{D}$ to Eq. (4.5), the continuity is satisfied:

$$\mathbf{D}\mathbf{u}' = -\mathbf{D}\mathbf{F}^{-1}\mathbf{G}p' = 0 \tag{4.6}$$

where letting $\mathbf{S} = -\mathbf{D}\mathbf{F}^{-1}\mathbf{G}$ and pressure-correction algorithm can be carried out via

$$\underbrace{\begin{bmatrix} \mathbf{F} & 0 \\ 0 & \mathbf{S} \end{bmatrix}}_{\tilde{\mathbf{A}}} \cdot \underbrace{\begin{bmatrix} \mathbf{u}^* \\ p' \end{bmatrix}}_{\tilde{\mathbf{W}}} = \underbrace{\begin{bmatrix} \mathbf{f} - \mathbf{G}p^* \\ \mathbf{D}\mathbf{u}' \end{bmatrix}}_{\tilde{\mathbf{b}}} \tag{4.7}$$

Thanks to the block-structure of matrix $\tilde{\mathbf{A}}$, this system $\tilde{\mathbf{A}}\tilde{\mathbf{W}} = \tilde{\mathbf{b}}$ can be solved via segregatedly computing two linear system momentum equations $\mathbf{F}\mathbf{u}^* = \mathbf{f} - \mathbf{G}p^*$ and pressure correction equation $-\mathbf{D}\mathbf{F}^{-1}\mathbf{G}p' = \mathbf{D}\mathbf{u}'$. For incompressible flow, $\mathbf{D}\mathbf{u} = 0$, and then

$$\mathbf{D}(\mathbf{u}^* + \mathbf{u}') = 0, \ \mathbf{D}\mathbf{u}' = -\mathbf{D}\mathbf{u}^* \tag{4.8}$$

Because the velocity correction value $\mathbf{u}'$ is unknown before the pressure correction value obtained, the right hand side of pressure correction equation needs to be replaced by $-\mathbf{D}\mathbf{u}^*$. Therefore, the fractional algorithm completes as:

- Momentum equation calculation for intermediate velocity $\mathbf{u}^*$:

$$\mathbf{F}\mathbf{u}^* = \mathbf{f} - \mathbf{G}p^*. \tag{4.9}$$

- Continuity equation calculation for pressure correction p':

$$\mathbf{S}p' = -\mathbf{D}\mathbf{u}^*. \tag{4.10}$$

- Correction step for velocity $\mathbf{u}$ and pressure p updating,

where the correction step is

$$\begin{aligned} \mathbf{u} &= \mathbf{u}^* + \mathbf{u}' = \mathbf{u}^* - \mathbf{F}^{-1}\mathbf{G}p' \\ p &= p^* + p' \end{aligned} \tag{4.11}$$

and primitive variables vector $\mathbf{W}$ can be expressed by left operator $\mathbf{L}$, intermediate variables vector $\tilde{\mathbf{W}}$ and initial pressure field:

$$\begin{bmatrix} \mathbf{u} \\ p \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{I} & -\mathbf{F}^{-1}\mathbf{G}p' \\ 0 & \mathbf{I} \end{bmatrix}}_{\mathbf{L}} \begin{bmatrix} \mathbf{u}^* \\ p' \end{bmatrix} + \begin{bmatrix} 0 \\ p^* \end{bmatrix} \tag{4.12}$$

So far the derivation of SIMPLE algorithm is done. The relationship between PSC and SIMPLE-family algorithms is not straightforward, however, the major characteristics are in common. From the view of prediction-correction as applied in SIMPLE algorithm, the original incompressible system can be expressed as:

$$\begin{bmatrix} \mathbf{F} & \mathbf{G} \\ \mathbf{D} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}^* + \mathbf{u}' \\ p^* + p' \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ 0 \end{bmatrix} \tag{4.13}$$

Let $\mathbf{W}^0 = \begin{bmatrix} 0 \\ p^* \end{bmatrix}$ and $\tilde{\mathbf{W}} = \begin{bmatrix} \mathbf{u}^* \\ p' \end{bmatrix}$. Primitive variables vector can be expressed as

$$\mathbf{W} = \mathbf{L}\tilde{\mathbf{W}} + \mathbf{W}^0 \tag{4.14}$$

where the left operator $\mathbf{L}$ is

$$\begin{bmatrix} \mathbf{I} & -\mathbf{F}^{-1}\mathbf{G} \\ 0 & \mathbf{I} \end{bmatrix} \tag{4.15}$$

The system $\mathbf{A}\mathbf{W} = \mathbf{b}$ can be transformed as:

$$\mathbf{A}(\mathbf{L}\tilde{\mathbf{W}} + \mathbf{W}^0) = \mathbf{b} \tag{4.16}$$

$$\mathbf{A}\mathbf{L}\tilde{\mathbf{W}} = \mathbf{b} - \mathbf{A}\mathbf{W}^0 = \begin{bmatrix} \mathbf{f} - \mathbf{G}p^* \\ 0 \end{bmatrix} \tag{4.17}$$

Let the right hand side equal $\tilde{\mathbf{b}}$. In order to solve saddle-point monolithic system

$\mathbf{AL}\tilde{\mathbf{W}} = \tilde{\mathbf{b}}$, right preconditioning (or postconditioning) matrix $\mathbf{P}$ can be applied

$$\mathbf{APP}^{-1}\mathbf{L}\tilde{\mathbf{W}} = \tilde{\mathbf{b}} \tag{4.18}$$

This preconditioner $\mathbf{P}$ can be decomposed in terms of matrix $\mathbf{A}$ and $\mathbf{P} = \mathbf{LM}^{-1}$, where $\mathbf{M}$ is

$$\mathbf{M} = \begin{bmatrix} \mathbf{F} & 0 \\ \mathbf{D} & \mathbf{S} \end{bmatrix} \tag{4.19}$$

The Schur complement $\mathbf{S}$ of block 0 of matrix $\mathbf{A}$ is $\mathbf{S} = 0 - \mathbf{DF}^{-1}\mathbf{G} = -\mathbf{DF}^{-1}\mathbf{G}$, which is the same with the coefficient matrix of pressure correction equation in Eq.(4.6). $\mathbf{M}$ can be further represented as two parts

$$\underbrace{\begin{bmatrix} \mathbf{F} & 0 \\ \mathbf{D} & \mathbf{S} \end{bmatrix}}_{\mathbf{M}} = \underbrace{\begin{bmatrix} \mathbf{F} & 0 \\ 0 & \mathbf{S} \end{bmatrix}}_{\tilde{\mathbf{A}}} + \underbrace{\begin{bmatrix} 0 & 0 \\ \mathbf{D} & 0 \end{bmatrix}}_{\mathbf{E}} \tag{4.20}$$

And then we can obtain

$$(\tilde{\mathbf{A}} + \mathbf{E})\tilde{\mathbf{W}} = \tilde{\mathbf{b}} \tag{4.21}$$

$$\tilde{\mathbf{A}}\tilde{\mathbf{W}} = \tilde{\mathbf{b}} - \mathbf{E}\tilde{\mathbf{W}} \tag{4.22}$$

The fractional equation of $\mathbf{u}^*$ from the system Eq.(4.22) is

$$\mathbf{Fu}^* = \mathbf{f} - \mathbf{G}p^* \tag{4.23}$$

and pressure correction equation comes as

$$\mathbf{S}p' = -\mathbf{Du}^* \tag{4.24}$$

Therefore a bridge between PSC and SIMPLE algorithm is built up based on different view: PSC is focused on the transformation of system matrix $\mathbf{A}$ through preconditioning method and SIMPLE algorithm comes from the prediction-correction idea. However, both of them have the same linearisation strategy like Picard fixed point iteration and the same coefficient linear matrices $\tilde{\mathbf{A}}$ including velocity system coefficient component $\mathbf{F}$ and pressure correction system coefficient component $\mathbf{S}$. These characteristics in common bring them to face similar stabilisation issues.

From the pressure Schur complement point of view, standard SIMPLE algorithm applies diagonal component of $\mathbf{F}$ to obtain its inverse approximation $\mathbf{F}^{-1}$. In order to improve the stability of the non-linear flow solver, coupling with the off-diagonal parts of $\mathbf{F}$ and $\mathbf{b}$ is achieved through "neighbour-correction" and "skewness correction" which enhances the discrete consistency of velocity and pressure and results

in better convergence, such as SIMPLEC and PISO algorithm. Compared to the fully coupled algorithms where the implicit scheme would require to solve a $4N \times 4N$ linear system (for 3 dimensional case), this family of methods only solves a sequence of loosely coupled $N \times N$ problems. Hence our first interest in stabilising the flow (primal) discretisation in the context of discrete adjoints should be to maximise the stability of these schemes while retaining its slim memory footprint. Of course, a fully coupled solver[116] would improve stability, however at a cost of significantly increased memory use.

4.2 Stabilisation strategies

According to the analysis of SIMPLE-like algorithms, there are some stabilisation methods for improving flow solver's performance discussed in this section.

4.2.1 Schur Complement

Reviewing implementation of SIMPLE-like algorithms, they generally can be carried as:

- Initializing velocity field and guessing pressure field;
- Calculating momentum equation in block system (4.22) $\mathbf{F}\mathbf{u}^* = \mathbf{f} - \mathbf{G}p^*$;
- Pressure Schur complement approximation: $\mathbf{S} \approx -\mathbf{D}\mathbf{F}^{-1}_{approx}\mathbf{G}$;
- Calculating pressure correction equation $\mathbf{S}p' = -\mathbf{D}\mathbf{u}^*$ with $\mathbf{S}$ obtained from last step;
- Evaluating velocity correction part $\mathbf{D}\mathbf{u}' = \mathbf{S}p'$;
- Correcting velocity and pressure.
- With updating velocity and pressure, going back to calculating momentum in step 2.

As discussed above, different PSC distinguishes SIMPLE, SIMPLEC and PISO algorithms. As we can see the most expensive part in PSC is calculating $\mathbf{F}^{-1}$, how to approximate it becomes essential to differentiate the different algorithms in SIMPLE-family. Meanwhile, there are two key places during the step-by-step to implement PSC: computing pressure correction linear system and evaluating velocity correction values; therefore, for PSC implementation $\mathbf{S}_{pres}$ is applied for pressure correction equation and $\mathbf{S}_{eva}$ is applied for velocity correction evaluation step. Generally speaking, if PSC only takes diagonal information of momentum matrix $\mathbf{F}$ or $\mathbf{F}^{-1}_{approx}$ has

only N components (where N is the cells' number, like SIMPLE and SIMPLEC), the same PSC are used $\mathbf{S}_{pres} = \mathbf{S}_{eva}$. However, when comes to approximate $\mathbf{F}^{-1}$ with keeping off-diagonal information, different implement of PSC shows different performance.

SIMPLE

For standard SIMPLE algorithm, the approximate inverse matrix $\mathbf{F}^{-1}$ is obtained via:

$$\mathbf{F}^{-1} \approx \begin{bmatrix} \frac{1}{a_{11}} & \cdots & \cdots & \cdots \\ \cdots & \frac{1}{a_{22}} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \frac{1}{a_{nn}} \end{bmatrix} \tag{4.25}$$

where a_{ii} is the diagonal component in matrix $\mathbf{F}$. The momentum matrix is diagonal dominant because of velocity under-relaxation factor applied on pole components as a consequence. This matrix formulation is equivalent with Eq. (2.19) in Chapter 2, where no neighbour cells' influence on pole cell for velocity correction part.

SIMPLEC[33] "Neighbour Correction"

As discussed in Chapter 2, velocity correction and pressure correction are balanced exactly as:

$$\phi_{i,I}^{'} = -\frac{\sum_J A_J^{\phi_i} \phi_{i,J}^{'}}{A_I^{\phi_i}} - \frac{1}{A_I^{\phi_i}} (\frac{\delta p^{'}}{\delta x_i})_I \Delta\Omega \tag{4.26}$$

where SIMPLE cancels the first term for simplification. Assume velocity correction $\phi_{i,I}^{'}$ can be approximated as wieghted average value of abtting cells' values:

$$\phi_{i,I}^{'} \approx -\frac{\sum_J A_J^{\phi_i} \phi_{i,J}^{'}}{\sum_J A_J^{\phi_i}} \tag{4.27}$$

And then the neighbour's influence can be approximated by transforming equation above as:

$$\sum_J A_J^{\phi_i} \phi_{i,J}^{'} \approx \phi_{i,I}^{'} \sum_J A_J^{\phi_i} \tag{4.28}$$

By introducing this neighbour's correction back to Eq. (4.26), relationship between velocity and pressure correction in SIMPLEC algorithm can be expressed as:

$$\phi_{i,I}^{'} \approx -\frac{1}{\sum_J A_J^{\phi_i} + A_I^{\phi_i}} (\frac{\delta p^{'}}{\delta x_i})_I \Delta\Omega \tag{4.29}$$

For SIMPLEC algorithm, the approximation inverse matrix $\mathbf{F}^{-1}$ is obtained in matrix formulation as:

$$\mathbf{F}^{-1} \approx \begin{bmatrix} \frac{1}{a_{11}+\sum a_{1j}} & \cdots & \cdots & \cdots \\ \cdots & \frac{1}{a_{22}+\sum a_{2j}} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \frac{1}{a_{nn}+\sum a_{nj}} \end{bmatrix} \tag{4.30}$$

where a_{ii} and a_{ij} are the diagonal and off-diagonal components in matrix $\mathbf{F}$.

Incomplete LU decomposition

In preconditioning techniques, inverse matrix is approximated widely used to constructure precondition matrix in order to accelerate linear system convergence. Incomplete LU (ILU) decomposition is widely used as efficient iterative method for linear sparse system and also works as preconditioner for linear solver based on Krylov subspace methods like conjugate gradiend, GMRES (the generalized minimum residual method) and other approaches [39, 120, 121]. The approximate inverse matrix for pressure Schur complement in SIMPLE-family algorithms can be built up via Incomplete LU decomposition.

The inverse matrix satisfies

$$\mathbf{F}\mathbf{F}^{-1} = \mathbf{I} \tag{4.31}$$

then it can be decomposed as series linear systems

$$\mathbf{F}X_1 = e_1; \mathbf{F}X_2 = e_2; \cdots; \mathbf{F}X_N = e_N \tag{4.32}$$

where $e_1, e_2, ..., e_N$ are unit vectors, and for e_i the ith component value is 1. $X_1, X_2, ..., X_N$ are column components of inverse matrix $\mathbf{F}^{-1}$. Here, ilu(0) [39] is carried out for approximate inverse matrix $\mathbf{F}^{-1}$, which means the sparsity pattern of $\mathbf{L}$ and $\mathbf{U}$ is chosen to be the same as the sparsity pattern of the original matrix $\mathbf{F}$:

$$\mathbf{F} \approx \mathbf{L}\mathbf{U} \tag{4.33}$$

and then let $\mathbf{L}\mathbf{U}(\mathbf{L}\mathbf{U})^{-1} = \mathbf{I}$, so the inverse matrix $\mathbf{F}^{-1}$ can be approximated as $(\mathbf{L}\mathbf{U})^{-1} = \mathbf{U}^{-1}\mathbf{L}^{-1}$. Therefore, this approximate inverse matrix $N \times N$ is calculated as:

```
Call ILU decomposition (F, L, U)
Do i=1,N
     ei=[0, ..., 1, ...,0]
     Call LU linear solver(L, U, e, xi)
```

```
    Assemble  F^{-1}=[.., xi,...]
End Do
```

Only diagonal components are stored for pressure Schur complement, and in that case no more structure modification is needed for continuity matrix set-up and correction step of velocity. However, for large scale grid case (e.g. N cells' case), the computation for pressure Schur complement is massive, as additional N linear systems need to be computed at each outer iteration and linear system scale is $N \times N$ that is the coefficient matrix dimension. Besides, momentum and pressure correction equations, the pressure Schur complement approximation via ILU decomposition method requires extra massive CPU time.

Sparse Approximate Inverse based on Frobenius Norm Minimization

As discussed above, ILU decomposition method pays the price of large CPU time because the global scale $N \times N$ linear system needs to be computed to obtain pressure Schur complement. Rather than calculating this scale system, Sparse Approximate Inverse (SAI) method is applied to build up local system for global sparse matrix $\mathbf{F}$ for inverse matrix approximation.

Let $\mathbf{M} \approx \mathbf{F}^{-1}$, and keep the same sparsity for M and calculate the constrained minimization problem:

$$min|\mathbf{I} - \mathbf{FM}|_F \tag{4.34}$$

where the subscript F stands for Frobenius. Similarly,

$$|\mathbf{I} - \mathbf{FM}|_F^2 = \sum_{i=1}^{N} |e_i - \mathbf{F}X_i|_2^2 \tag{4.35}$$

which leads to solving N independent linear least-squares problems, subject to the sparsity constraints. And QR decomposition (also called as QR factorization) is used to solver the linear least-squares problem. For each column of matrix $\mathbf{M}_k$ the non-zero components number n_k is the same with counterpart row of $\mathbf{F}$, and local system can be built as:

$$\mathbf{A}_{n_k \times n_k} \mathbf{M}_k = e_k \tag{4.36}$$

where values in $\mathbf{A}_{n_k \times n_k}$ is picked up from sparse matrix $\mathbf{F}$. Here the implementation steps are

```
Do i=1,N
  Build up local system A(n_k, n_k) and right hand side r_k
  Call QR decomposition (A, Q, R)
  Calculate right hand side Q^{T}r_k for QR decomposition
```

```
  Call backsubstitution (R,Q^{T}r_k,M_k)
  Store M_k in F^{-1}_{approx}
End Do
```

where QR factorization uses Householder transformations[122]. Once approximation of $\mathbf{M}$ obtained, only the diagonal components are taken for pressure correction equation. For correction step off-diagonal data is used. Based on this implementation, convergence speed is not sensitive to the choice of pressure under-relaxation any more. With the the velocity under-relaxation, arbitrary pressure under-relaxation in work space keeps the same convergence speed. Numerical tests also show that for given velocity under-relaxation 0.7 on 3D S-bend duct case, SAI scheme gives larger work space for pressure under-relaxation [0.1 : 0.5] rather than [0.1 : 0.3] for standard SIMPLE, but less than work space of SIMPLEC [0.1 : 1.0].

4.2.2 Semi-coupled Jacobian

The proposed method is a semi-coupled implicit solver with a block-Jacobi representation of the advection matrix $\mathbf{F}^{-1}$ which provides a better approximation to the pressure Schur complement S. In FVM based SIMPLE-family algorithms, the velocity components u, v and w have the same discretilized coefficient matrix A_I, which brings the benefit of small memory required. Segregated calculation of each linear system is carried out through respective preconditioning and iterative solvers. The semi-coupled system is built up via solving velocity components together. This block Jacobi for momentum (e.g. 3 dimension case)

$$\mathbf{F} = \begin{bmatrix} \mathbf{A}_u & \mathbf{A}_{uv,I} & \mathbf{A}_{uw,I} \\ \mathbf{A}_{vu,I} & \mathbf{A}_v & \mathbf{A}_{vw,I} \\ \mathbf{A}_{wu,I} & \mathbf{A}_{wv,I} & \mathbf{A}_w \end{bmatrix} \tag{4.37}$$

which gives a more sufficient preconditioning procedure and this implementation strategy improves the stability. For 3 dimensional cases, semi-coupled solver theoretically requires nearly half memory of fully-coupled algorithm, as $3N \times 3N$ comparing to $4N \times 4N$. Submatricies $\mathbf{A}_i$ for $i = u, v, w$, containing elements $A_{i,I}$ and $A_{i,J}$, stand for coefficient matrix of ith component of velocity with the flux contribution to cell I and the same velocity component with the flux contribution to neighbour cell J. Off diagonal submatriceis $\mathbf{A}_{ij,I}$ for $i = u, v, w; j = u, v, w$ and $i \neq j$ couple the velocity components u, v and w with the flux the contribution to their cell where they exist. For internal cells, the coefficient Jacobian matrix for velocity can be calculated:

$$\begin{gathered} A_{u,I} = \sum A_{u,J}; \quad A_{v,I} = \sum A_{v,J}; \quad A_{w,I} = \sum A_{w,J} \\ A_{uv,I} = 0; \quad A_{uv,I} = 0; \quad A_{uv,I} = 0 \end{gathered} \tag{4.38}$$

For boundary cells, non permeable wall's shear stress is

$$\tau_b = \frac{\mathbf{u}_{t,I} - \mathbf{u}_b}{\mathbf{d}_{Ib} \cdot \mathbf{n}_b} \tag{4.39}$$

the boundary tangential velocity :

$$\mathbf{u}_{t,I} = \mathbf{u}_I - (\mathbf{u}_I \cdot \mathbf{n}_b)\mathbf{n}_b \tag{4.40}$$

where $\mathbf{d}_{Ib}$ is the displacement vector pointing from boundary face center to boundary cell centroid I. The boundary diffusion flux contribution to cell I:

$$\mathbf{F}_b = -\tau_b \mathbf{S}_b \tag{4.41}$$

For segregated pressure-based algorithm, the boundary viscous flux is calculated explicitly and coupled at right hand side as source term, as in this way the identical system matrices are applied for different velocity components. However, for skew boundary cell, the geometric correction term needs to be added via deferred-correction method, and that means the correction of fluxes is deferred for current values. Semi-coupled algorithm couples velocity components implicitly with more accurate diffusion flux calculation:

$$\begin{aligned}
F_{u,b} &= -\frac{\Gamma \Delta S}{\mathbf{d}_{Ib} \cdot \mathbf{n}_b}((1 - n_x^2)u_I - n_x n_y v_I - n_x n_z w_I - u_b) \quad \text{for x direction} \\
F_{u,b} &= -\frac{\Gamma \Delta S}{\mathbf{d}_{Ib} \cdot \mathbf{n}_b}((1 - n_x^2)v_I - n_y n_x u_I - n_y n_z w_I - v_b) \quad \text{for y direction} \\
F_{u,b} &= -\frac{\Gamma \Delta S}{\mathbf{d}_{Ib} \cdot \mathbf{n}_b}((1 - n_x^2)w_I - n_z n_x u_I - n_x n_y v_I - w_b) \quad \text{for z direction}
\end{aligned} \tag{4.42}$$

and then the boundary flux contribution to boundary cell can be added to coefficient matrix as:

$$\begin{aligned}
A_{u,I} &:= A_{u,I} + \frac{\Gamma \Delta S}{\mathbf{d}_{Ib} \cdot \mathbf{n}_b}(1 - n_x^2) \\
A_{v,I} &:= A_{v,I} + \frac{\Gamma \Delta S}{\mathbf{d}_{Ib} \cdot \mathbf{n}_b}(1 - n_y^2) \\
A_{w,I} &:= A_{w,I} + \frac{\Gamma \Delta S}{\mathbf{d}_{Ib} \cdot \mathbf{n}_b}(1 - n_z^2)
\end{aligned} \tag{4.43}$$

$$
\begin{aligned}
A_{uv,I} &:= A_{uv,I} - \frac{\Gamma \Delta S}{\mathbf{d}_{Ib} \cdot \mathbf{n}_b} n_x n_y \\
A_{uw,I} &:= A_{uw,I} - \frac{\Gamma \Delta S}{\mathbf{d}_{Ib} \cdot \mathbf{n}_b} n_x n_z \\
A_{vu,I} &:= A_{vu,I} - \frac{\Gamma \Delta S}{\mathbf{d}_{Ib} \cdot \mathbf{n}_b} n_y n_x \\
A_{vw,I} &:= A_{vw,I} - \frac{\Gamma \Delta S}{\mathbf{d}_{Ib} \cdot \mathbf{n}_b} n_y n_z \\
A_{wu,I} &:= A_{wu,I} - \frac{\Gamma \Delta S}{\mathbf{d}_{Ib} \cdot \mathbf{n}_b} n_z n_x \\
A_{wv,I} &:= A_{wv,I} - \frac{\Gamma \Delta S}{\mathbf{d}_{Ib} \cdot \mathbf{n}_b} n_w n_y
\end{aligned}
\tag{4.44}
$$

Numerical tests show that off-diagonal coupling flux calculated explicitly brings better stability. And partly deferred-correction term is moved to right hand side:

$$
\begin{aligned}
b_{u,I} &:= b_{u,I} + \left(\frac{\Gamma \Delta S}{\mathbf{d}_{Ib} \cdot \mathbf{n}_b} n_x n_y \right) v_I^{m-1} + \left(\frac{\Gamma \Delta S}{\mathbf{d}_{Ib} \cdot \mathbf{n}_b} n_x n_z \right) w_I^{m-1} \\
b_{v,I} &:= b_{v,I} + \left(\frac{\Gamma \Delta S}{\mathbf{d}_{Ib} \cdot \mathbf{n}_b} n_y n_x \right) u_I^{m-1} + \left(\frac{\Gamma \Delta S}{\mathbf{d}_{Ib} \cdot \mathbf{n}_b} n_y n_z \right) w_I^{m-1} \\
b_{w,I} &:= b_{w,I} + \left(\frac{\Gamma \Delta S}{\mathbf{d}_{Ib} \cdot \mathbf{n}_b} n_z n_x \right) u_I^{m-1} + \left(\frac{\Gamma \Delta S}{\mathbf{d}_{Ib} \cdot \mathbf{n}_b} n_x n_z \right) v_I^{m-1}
\end{aligned}
\tag{4.45}
$$

Pressure Schur Complement for Semi-coupled Algorithm Based on semi-coupled Jacobian velocity system, the pressure correction flux are discretilised as:

$$
\int_{\Delta\Omega} -\mathbf{D}\mathbf{F}^{-1}\mathbf{G}p' d\Omega = \sum -\mathbf{F}_{k,n}^{-1} \nabla p'_k \cdot \mathbf{S} \tag{4.46}
$$

where $\mathbf{F}_k^{-1}$ is interpolated face value. For standard approximation of inverse matrix of $\mathbf{F}$, the face interpolated inverse matrix is:

$$
\mathbf{F}_k^{-1} = \begin{bmatrix} F_{u,k}^{-1} & 0 & 0 \\ 0 & F_{v,k}^{-1} & 0 \\ 0 & 0 & F_{w,k}^{-1} \end{bmatrix} \tag{4.47}
$$

where diagonal components are calculated as:

$$
\begin{aligned}
F_{u,k}^{-1} &= \lambda_k \Delta\Omega_I A_{u,I}^{-1} + (1-\lambda)\Delta\Omega_J A_{u,J}^{-1} \\
F_{v,k}^{-1} &= \lambda_k \Delta\Omega_I A_{v,I}^{-1} + (1-\lambda)\Delta\Omega_J A_{v,J}^{-1} \\
F_{w,k}^{-1} &= \lambda_k \Delta\Omega_I A_{w,I}^{-1} + (1-\lambda)\Delta\Omega_J A_{w,J}^{-1}
\end{aligned}
\tag{4.48}
$$

$$
\begin{aligned}
A_{p',I} &= -\sum \frac{\mathbf{F}_k^{-1}\mathbf{nS}}{\mathbf{d}\cdot\mathbf{n}} \\
&= -\sum \frac{(\mathbf{F}_{k,n}^{-1})\cdot\Delta S}{\mathbf{d}\cdot\mathbf{n}} \\
&= -\sum(\lambda_k\Delta\Omega_I A_{u,I}^{-1} + (1-\lambda)\Delta\Omega_J A_{u,J}^{-1})n_x^2 \\
&+ (\lambda_k\Delta\Omega_I A_{v,I}^{-1} + (1-\lambda)\Delta\Omega_J A_{v,J}^{-1})n_y^2 \\
&+ (\lambda_k\Delta\Omega_I A_{w,I}^{-1} + (1-\lambda)\Delta\Omega_J A_{w,J}^{-1})n_z^2
\end{aligned}
\tag{4.49}
$$

$$
\begin{aligned}
A_{p',J} &= \frac{\mathbf{F}_k^{-1}\mathbf{nS}}{\mathbf{d}\cdot\mathbf{n}} \\
&= (\lambda_k\Delta\Omega_I A_{u,I}^{-1} + (1-\lambda)\Delta\Omega_J A_{u,J}^{-1})n_x^2 \\
&+ (\lambda_k\Delta\Omega_I A_{v,I}^{-1} + (1-\lambda)\Delta\Omega_J A_{v,J}^{-1})n_y^2 \\
&+ (\lambda_k\Delta\Omega_I A_{w,I}^{-1} + (1-\lambda)\Delta\Omega_J A_{w,J}^{-1})n_z^2
\end{aligned}
\tag{4.50}
$$

where $A_{i,I}$ is the diagonal components of inverse matrix $\mathbf{F}^{-1}$. For standard SIMPLE family algorithm, the component from momentum equation (also the one in PSC):

$$
\overline{\frac{\Delta\Omega}{A_I^\phi|_k}} \tag{4.51}
$$

where $\overline{\frac{1}{A_I^\phi}}$ is interpolated face value. Because $\frac{1}{A_I^\phi} = A_{u,I}^{-1} = A_{v,I}^{-1} = A_{w,I}^{-1}$, an then linear interpolated face value is:

$$
\begin{aligned}
\overline{\frac{1}{A_I^\phi}} &= \lambda_k A_{u,I}^{-1} + (1-\lambda_k)A_{u,J}^{-1} \\
&= \lambda_k A_{v,I}^{-1} + (1-\lambda_k)A_{v,J}^{-1} \\
&= \lambda_k A_{w,I}^{-1} + (1-\lambda_k)A_{w,J}^{-1}
\end{aligned}
\tag{4.52}
$$

and interpolated face volume is obtained:

$$
\overline{\Delta\Omega_k} = \Delta S\Delta x_{IJ} \tag{4.53}
$$

Consequently, the $\mathbf{F}$ inverse matrix value for each control volume I is calculated as constant scalar, and pressure correction flux:

$$
\begin{aligned}
f_k &= -\rho\Delta\Omega\Delta S)_k\overline{\frac{1}{A_{u,I}}}_k\left(\frac{\delta p'}{\delta n}\right)_k \\
&= -\rho\Delta S_k\Delta x_{IJ}\Delta S_k(\lambda_k A_{u,I}^{-1} + (1-\lambda_k)A_{u,J}^{-1})\frac{p'_J - p'_I}{\Delta x_{IJ}} \\
&\approx -\rho\Delta S_k^2\lambda_k\frac{1}{A_{u,I}} + (1-\lambda)\frac{1}{A_{u,J}}p'_J - p'_I
\end{aligned}
\tag{4.54}
$$

Being different from standard SIMPLE-like algorithms, the matrix inverse com-

ponent in pressure Schur complement is applied in pressure correction equation via matrix formula:

$$\mathbf{F}_I^{-1} = \begin{bmatrix} \Delta\Omega_I A_{u,I}^{-1} & 0 & 0 \\ 0 & \Delta_I A_{v,I}^{-1} & 0 \\ 0 & 0 & \Delta_I A_{w,I}^{-1} \end{bmatrix} \tag{4.55}$$

and the face interpolated $\mathbf{F}_k^{-1}$ is evaluated as:

$$\mathbf{F}_k^{-1} = \lambda_k \mathbf{F}_I^{-1} + (1 - \lambda_k)\mathbf{F}_J^{-1} \tag{4.56}$$

where volume weights is used for face value interpolation rather than distance weights:

$$\lambda_k = \frac{\Delta\Omega_I}{\Delta\Omega_I + \Delta\Omega_J} \tag{4.57}$$

This implementation keeps a compact stencil, and similar implementation proposed [123] recently shows stabilisation effect of removing convergence oscillation, but its formula is derived only for structure grid and velocity components are computed segregatedly using the same system matrix. The implementation here gives general formulation for co-allocated face-based grid with both normal segregated algorithm and semi-coupled algorithm. PSC implementation proposed here is not limit to standard SIMPLE PSC inverse matrix approximation and can be easily extended to SIMPLEC inspired PSC and other PSC methods. In PSC method mentioned above, the inverse matrix keeps off-diagonal data in its approximation. Only diagonal values are applied to calculate coefficient value in pressure correction equation rather the exact approximate inverse matrix with off-diagonal data. This simplification is more stable and requires easier correction step implementation, which is tested by numerical experience.

4.3 Skewness correction

In order to enhance the solver robustness for skew grid, 2-step correction is applied for semi-coupled solver. It is assumed that the second pressure correction equation leads to mass conservation, so the net outward flux of face velocity is supposed to equal to mass residual of control volume:

$$\frac{1}{A_I^{\phi_i} + \sum_J A_J^{\phi_i}} (\nabla p''_{i,I}) \Delta\Omega = \Delta m'. \tag{4.58}$$

where $\frac{1}{A_I^{\phi_i} + \sum_J A_J^{\phi_i}}$ stands for the approximation of the $\mathbf{F}^{-1}$ in pressure Schur complement in algebra formula, and we can find that neighbouring cells' influence is also

considered for second correction if PSC is chosen among SIMPLEC, ILU and SAI. In that case, velocity and pressure are corrected via:

$$p_I^m = p_I^* + \alpha_p p_I' + p_I''; \tag{4.59}$$

$$\phi_{i,I}^m = \phi_{i,I}^* + \phi_{i,I}' + \phi_{i,I}''; \tag{4.60}$$

Meanwhile, the control volume face normal gradient calculation determines robustness with respect to grid quality. For implicit algorithms of incompressible flow, the diffusion and pressure correction terms based on Green theorem are transformed in term of face normal gradient. The diffusive flux scheme is derived using deferred-correction method, so correction term for skew mesh is implemented explicitly via:

$$F_f^d = \Gamma \Delta S \frac{\partial \phi}{\partial n} \approx \frac{\Gamma \Delta S}{\delta}(\phi_p - \phi_n) + F_c^d \tag{4.61}$$

F_c^d means correction part composed by more accurate explicit schemes. Improved deferred correction (IDC) scheme[124, 125] is implemented to improve the accuracy of face normal gradient:

$$F_c^d = \Gamma \Delta S (\nabla \phi)_f \cdot \beta \boldsymbol{\tau} \tag{4.62}$$

where $\boldsymbol{\tau}$ is unit face tangent vector, $\beta = tan(\theta_k)$ and θ_k is the angle between face k unit normal vector $\mathbf{n}$ and neighbour cells' centroids connection unit vector $\mathbf{i}_\xi$, shown in Fig. 4.1. And face normal vector can be expressed as:

$$\mathbf{n}_k = \beta \boldsymbol{\tau} + \alpha \mathbf{i}_\xi. \tag{4.63}$$

Normally in cell-centred FVM implementation, the $\mathbf{n}_k$ is stored at each face and

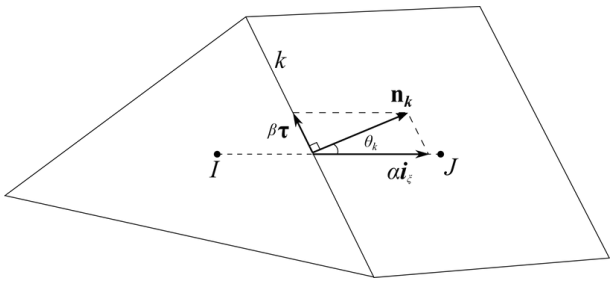


Figure 4.1: Face normal vector $\mathbf{n}_k$ decomposition in IDC

vector $\mathbf{i}_\xi$ is easy to calculated. Therefore, the correction is implemented as:

$$F_c^d = \Gamma \Delta S (\nabla \phi)_f \cdot (\mathbf{n}_k - \alpha \mathbf{i}_\xi) \tag{4.64}$$

where $\alpha = \frac{1}{cos(\theta)}$.

Skewness corrections via solving an additional pressure Poisson equations[111]

and improved deferred scheme lead to accurate gradient calculation when the mesh orthogonality is poor as shown in Fig. 4.2. And standard SIMPLE and SIMPLEC algorithms fail to converge for this lid-driven cavity case only if extreme conservative parameters are set. With default settings in Tab. 4.1, convergence behaviour shows in Fig. 4.3.

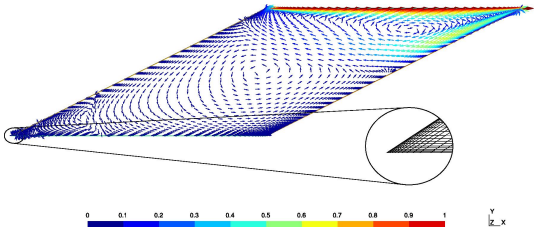


Figure 4.2: Cavity case on skew mesh via Semi-coupled solver

Table 4.1: Parameter settings for skew cavity case at Re=1000

Algorithm	α_u	α_p	Scheme	CBL	2nd Correction	Linear solver
SIMPLE	0.7	0.3	Min-Mod	0.55	-	Bi-CGSTAB/CG
SIMPLEC	0.7	1.0	Min-Mod	0.55	-	Bi-CGSTAB/CG
SEMI-COUPLED	0.7	1.0	Min-Mod	0.55	1 step	Bi-CGSTAB/CG

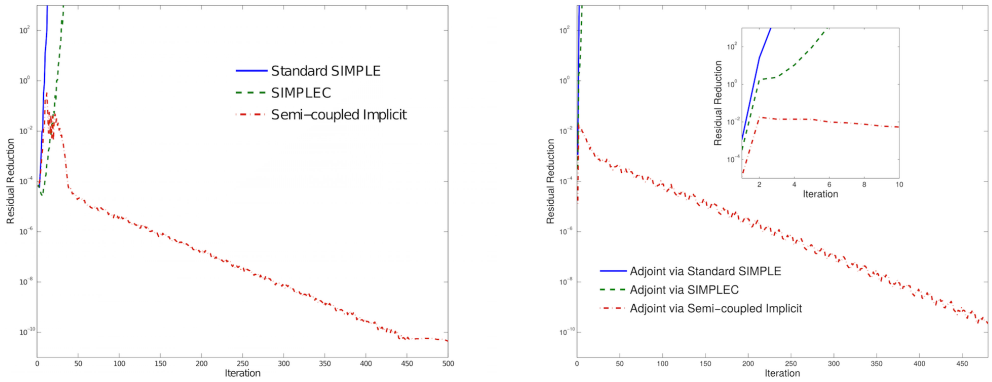


Figure 4.3: Convergence comparison among SIMPLE, SIMPLEC and Semi-coupled implicit solvers

Numerical experience shows that a more accurate gradient evaluation method used for pressure correction gradient stabilises the flow as well. For pressure correction algorithms based on the FVM, the pressure gradient in the discrete momentum equation is presented as source term, which is put on the right hand side:

$$A_I^{\phi_i}\phi_{i,I} + \sum_J A_J^{\phi_i}\phi_{i,J} = -\nabla p_{i,I}\Delta\Omega \tag{4.65}$$

where ϕ_i is a velocity component and A is the coefficient containing convection and

diffusion information. This spatial discrete equation is driven by the pressure term on the r.h.s.. In addition, the velocity correction values are determined by the pressure correction gradient field during the correction procedure:

$$\phi'_{i,I} = -\frac{1}{A_I^{\phi_i}}(\nabla p'_{i,I})\Delta\Omega \tag{4.66}$$

therefore, the gradient evaluation methods affect the accuracy and stability for the family of pressure correction algorithms.

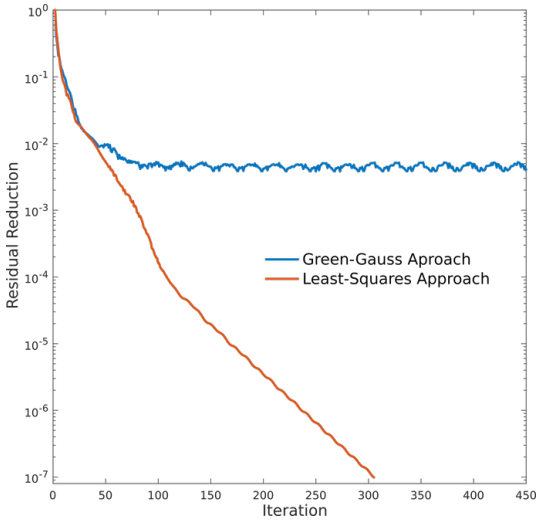


Figure 4.4: Convergence comparison between Green-Gauss and Least-Squares approaches for gradient evaluation

Test case on 3D S-bend duct shows the convergence curves obtained from Green-Gauss and Least-Squares approaches, seen in Fig. 4.4. Least-Squares approach shows strong stabilisation effect for this case. However, in a wide range of application, Least-Square approach doesn't perform robustly and diverges often.

4.4 Algebraic Multi-grid Technique from HYPRE

The linear systems with system matrices $\mathbf{A}_i$ and $\mathbf{S}$ for momentum in Eq. (4.26) and pressure in Eq. (4.6), respectively, are of similar size, $N \times N$. So at first glance, one could expect similar cost in solving either system, with the cost being proportional to N^3[126] for typical numerical methods. And indeed on rather coarse grids this is the case. However, the coefficients in the system matrices reflect the flow physics, and the flow physics are very different. The momentum equation Eq.(2.9) is hyperbolic. It describes transport along a streamline, a very local effect that strongly couples immediately neighbouring cells. Coupling along the streamline is essentially a one-dimensional phenomenon. The number of nodes along a streamline, for domains that are roughly square in the number of cells in each direction, in the order of

$M = \sqrt{N}$ in 2-D and $M = \sqrt[3]{N}$ in 3-D. The cost of solving these systems is $O(M^3)$, hence of order $M^3 = (\sqrt{N})^3 = N^{\frac{3}{2}}$ in 2-D and of $M^3 = (\sqrt[3]{N})^3 = N$ in 3-D. Hence the increase in the number of iterations to converge the system with increasing grid size is very moderate. The pressure correction is elliptic and describes the relaxation of the pressure field after infinitely long time compared to the flow speed, since the incompressible equ. assume infinitely large sound speed. This means that firstly, all cells are coupled to all cells, and secondly, the strength of the coupling does not decay as rapidly with distance from a given cell as it does in the momentum equation. Hence the cost of solving the system will be of order N^3, which grows dramatically with increasing mesh size.

In addition, by profiling the flow solver, when tolerances of the inner iterations for momentum and continuity equations are both 10^{-6}, it is shown that percentage relative to total CPU run-time of GPDE (3-D S-bend case on 47k mesh) taken by the CGSTAB solver is 8.03; while the time percentage taken by BICGSTAB solver is 0.93, which is much less than that for CGSTAB solver. It indicates that for pressure equation solution, it needs much more time consumption. And reducing the cost of pressure computation will save CPU time to a large extent in the computation of flow field. Consequently, the improvement of pressure solver is meaningful and important. Solving pressure Poisson equation is the most time-consuming part of the computation on larger meshes, which is the bottleneck in incompressible fluid computation. In industrial applications, the mesh sizes are usually large. When it comes to a whole vehicle shape optimisation, the size of the grid is often tens of millions of grid points, which requires highly efficient linear solvers to calculate the flow field.

HYPRE short for high performance pre-conditioners, which is a software library of high performance preconditioners and solvers for the solution of large, sparse linear systems of equations[127]. *HYPRE* contains several families of preconditioner algorithms focused on the scalable solution of very large sparse linear systems. Meanwhile, it provides the most commonly used Krylov-based iterative methods such as GMRES and preconditioned BICG solvers for asymmetrical systems and conjugate gradient methods for symmetric matrices. The user needs to select a single Krylov method and a single preconditioner for one linear algebra equation system according to the particular flow field simulation.

Among them the algebraic multi grid(AMG) method is considered as key technique for solving pressure correction Poisson equation. In contrast to the geometric multi-grid methods, which include element-based techniques [128] or agglomeration multi-grid methods [129], algebraic multi-grid (AMG) [130] does not require a given problem to be defined on a grid but rather operates directly on a linear, possibly

sparse algebraic equation:

$$\mathbf{A}x = \mathbf{b} \quad or \quad \sum_{j=1}^{n} a_{ij}x_j = b_i \quad (i = 1, 2, ..., n). \tag{4.67}$$

where n is the number of equations. For geometric multi-grid methods, equation systems are built on the different scale of grids (from coarse grid to fine grid), which takes the advantage of the rapid convergence on coarse-grid and keep certain accuracy via error correction from fine-grid.

This multi-grid method is also called as multi level method, as iterations at different levels of grid are carried out as following steps:

1. Perform enough iterations on the fine grid to remove high-frequency errors;
2. Compute the fine-grid residual;
3. Interpolate the fine-grid residual to the next coarser grid ("restriction");
4. Compute the residual of the restricted solution and add to a source term (which zeroes out effects from representing the fine grid solution on the coarse grid);
5. Add the restriction of the fine grid residual to the source term (which now drives the evolution of the coarse grid solution, since the initial coarse-grid residual has been subtracted out);
6. Perform iterations on the coarse grid;
7. Interpolate the difference between initial and iterated coarse grid solution to the fine grid ("prolongation");
8. Correct the fine grid solution with the prolongated coarse-grid correction.

If one replaces the terms grids and grid points by sets of variables and single variables respectively, one can describe AMG in formally the same way as a geometric multi-grid method [130]. In AMG, the multi 'grids' are generated based on the varibles' value dependency from the given system matrix. It means that AMG can work as a "black box" and fed with linear systems. It can be directly applied for linear equation systems without geometry information. Consequently, AMG can be treated as a dependent "plug-in" solver, and directly employed to efficiently solve various types of partial differential equations discretized on unstructured meshes in more efficient way [131]. In *HYPRE*, AMG named as BoomerAMG can be used as both preconditioner and standalone solver.

4.5 Validation and results discussion

4.5.1 Linear solver validation

Before linking the prospective linear solvers from *HYPRE* to GPDE, the validation case is conducted. As mentioned above, AMG technique can be used as linear solver alone and preconditioner for other method, so BoomerAMG and BoomerAMG-pre PCG solvers are compared, where PCG stands for preconditioned CG solver.

Comparison between BoomerAMG and BoomerAMG-pre PCG Given function

$$u(x, y) = x + y, \tag{4.68}$$

its Poisson equation is

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{4.69}$$

with computational domain is rectangle area Ω :$\{0 < x < 1, 0 < y < 1\}$. The boundary conditions at four sides are $u|_{y=0} = x$, $u|_{x=1} = y + 1$, $u|_{y=1} = x + 1$ and $u|_{x=0} = y$. Finite difference method is applied for this test case:

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} + o(\Delta x^2) \tag{4.70}$$

Numerical solution on 500×500 grid is shown in Fig. 4.5. Both solvers converge this linear system to an absolute residual as 10^{-7} and achieve the correct solution. Parameter settings are applied for both BoomerAMG used as solver and

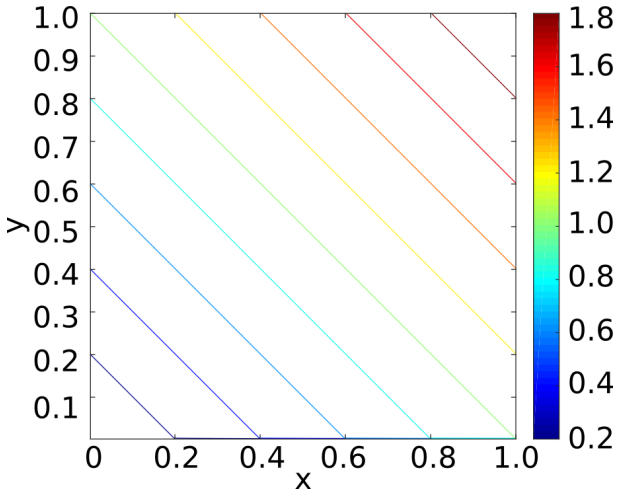


Figure 4.5: Solution of 2 dimensional Poisson equation test case

BoomerAMG-pre for PCG solver.

The comparison of absolute error distribution between two solvers is shown in Fig. 4.6. The same scale is used for both distribution graphs, and smaller error

Table 4.2: Parameter settings for algebraic multi grid solver and preconditioner

Method	Relaxation Type	Maximum Number of Cycle	Tolerance	Cycle Type
BoomerAMG	Hybrid Symmetric Gauss-Seidel	20	10^{-7}	V
BoomerAMG Pre	Hybrid Symmetric Gauss-Seidel	1	0.0	V

achieved from PCG solver with AMG-preconditioning is clearly illustrated. The order of magnitude of error from BoomerAMG solver is 10^{-7} and that from PCG solver with BoomerAMG solver is 10^{-8}.

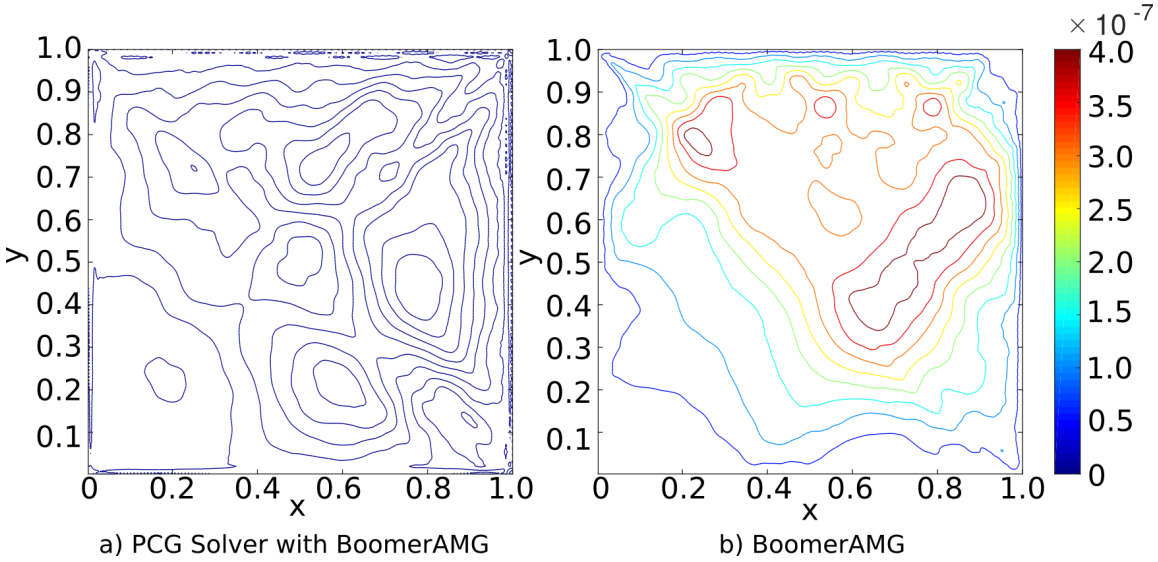


Figure 4.6: Error distribution comparison between a) PCG solver with AMG preconditioner and b) AMG Solver

Performance of difference coarsen types for BoomerAMG and PCG with BoomerAMG preconditioned solvers is listed in Appendix B.1. And AMG technique gives almost mesh-independent iteration numbers and CPU time for each node. PCG with BoomerAMG preconditioned solver use less iteration numbers compared with BoomerAMG, and the converged solution has smaller error. PCG with BoomerAMG preconditioned solver shows slightly superiority on the linear Poisson system.

4.5.2 Flow Solver Validation

Test case is Re=1000 bench mark lid-driven cavity flow. SIMPLE, SIMPLEC and Semi-Coupled solvers are carried on eight types of meshes, shown in Fig. 4.7. The mesh density for this Reynold number brings the numerical instability rather than physical one [132], because these grids are not fine enough to express detail physical unstable phenomena. For our test cases, based on the same grid, the performance of convergence shows the stability properties of each algorithm and the numerical instability damping effects of proposed solver's implementation.

For SIMPLE solver working on orthogonal grid and non-orthogonal grid but with-

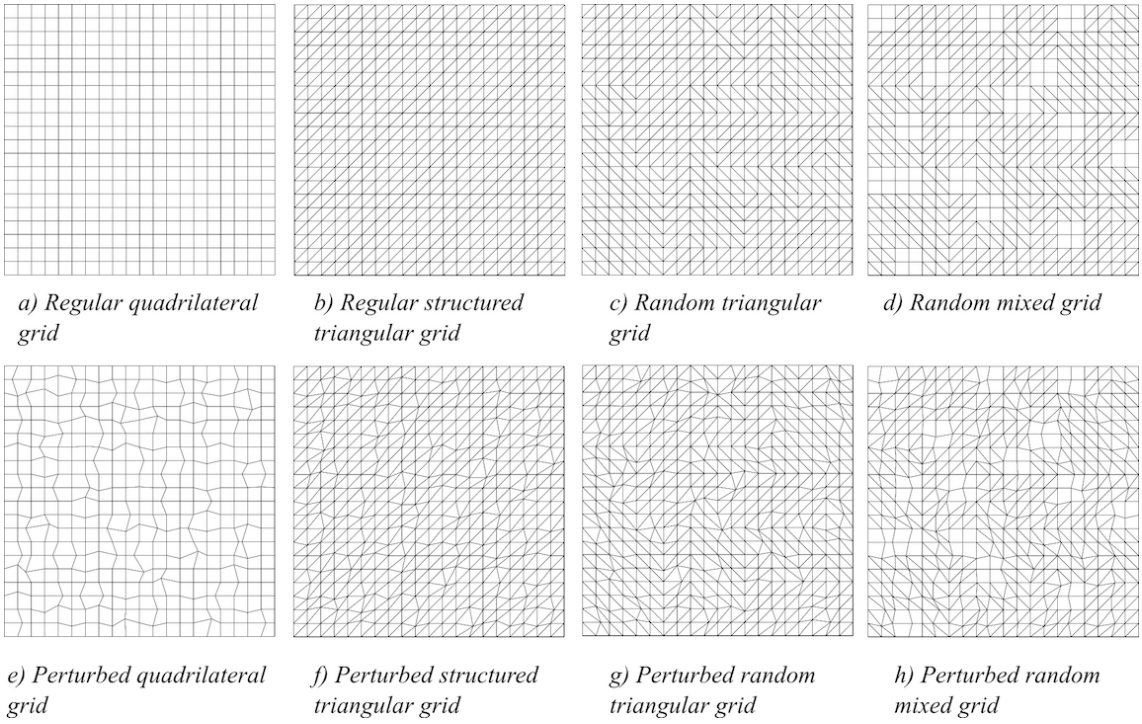


Figure 4.7: Types of grids prepared for 2D lid-driven cavity viscous flow

out serious skewness, the relationship between correction relaxation factors of velocity α_u and pressure α_p is suggested to satisfy:

$$\alpha_u + \alpha_p = c \tag{4.71}$$

where c is in the data range from 1 to 1.1 [133, 134]; and α_u is proposed in the range from 0.7 to 0.8. While for SIMPLEC algorithm the pressure under-relaxation can be set as 1.0 theoretically, and only for skew grid this value needs to be reduced. And semi-coupled applies the same approximation method for component in pressure Schur complement with SIMPLEC, so pressure under-relaxation for semi-coupled solver can be set as the same as SIMPLEC solver. That means that compared with standard SIMPLE solver, SIMPLEC and semi-coupled solvers have wider range for parameter setting and robustness on varied types of meshes. Moreover, an explicit correction method relying on time step multiple[135] was tested in SIMPLEC and PISO, which can reduce the calculation time based on the same convergence criteria. Therefore, taking the advantage of the balance between mass continuity and momentum conservation will be helpful for the convergence. Therefore, test cases' parameter settings are optimal as listed in Tab. 4.3:

Test is carried out based on the similar optimal parameters for each algorithm in order to achieve reasonable comparison. Velocity fields obtained on each types of grids are shown in Fig. 4.8. As we can see from the solution fields, the primary and secondary vortices appear on the same position for all types of mesh.

Table 4.3: Optimal stability parameter settings for SIMPLE, SIMPLEC and Semi-Coupled solvers

Algorithm	α_u	α_p	Scheme	cbl	2nd Correction	Linear solver
SIMPLE	0.7	0.3	Min-Mod	1.0	-	Bi-CGSTAB/CG
SIMPLEC	0.7	1.0	Min-Mod	1.0	-	Bi-CGSTAB/CG
SEMI-COUPLED	0.7	1.0	Min-Mod	1.0	1 step	Bi-CGSTAB/CG

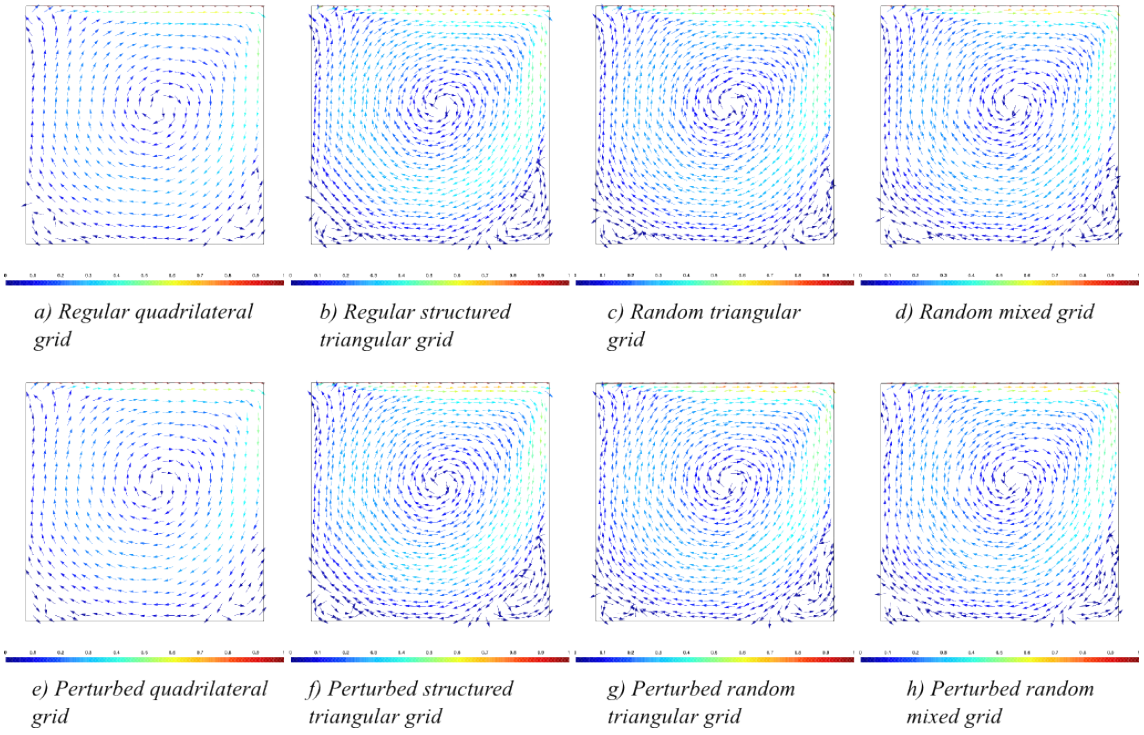


Figure 4.8: Types of grids prepared for 2D lid-driven cavity viscous flow

Table 4.4: Iteration numbers comparison among SIMPLE, SIMPLEC and Semi-Coupled solvers

Grid type	Cell number	SIMPLE	SIMPLEC	Semi-Coupled
		Percentage (Iterations)	Percentage	Percentage
Regular quadrilateral	400	100% (442)	87.33%	58.37%
Regular structured triangular	800	100% (511)	98.49%	98.15%
Random triangular	800	100% (590)	95.59%	92.03%
Radom mixed	736	100% (542)	94.28%	88.56%
Perturbed quadrilateral	400	100% (316)	93.63%	87.97%
Perturbed structured triangular	800	100% (573)	98.42%	97.90%
Perturbed random triangular	800	100% (635)	94.80%	90.55%
Perturbed random mixed	736	100% (575)	96.52%	82.95%

The iteration numbers and CPU time are compared and listed in Tab. 4.4 and Tab. 4.5. According to results of iteration percentage comparison, optimal setting of under-relaxation lets SIMPLE and SIMPLEC solvers take similar steps, and SIMPLEC solver converges slightly faster than standard SIMPLE; semi-coupled solver

Table 4.5: CPU time comparison among SIMPLE, SIMPLEC and Semi-Coupled solvers

Grid type	Cell number	SIMPLE	SIMPLEC	Semi-Coupled
		Percentage	Percentage	Percentage
Regular quadrilateral	400	100%	99.70%	65.22%
Regular structured triangular	800	100%	98.56%	97.88%
Random triangular	800	100%	114.7%	145.2%
Random mixed	736	100%	82.84%	106.2%
Perturbed quadrilateral	400	100%	88.02%	134.9%
Perturbed structured triangular	800	100%	106.7%	124.7%
Perturbed random triangular	800	100%	61.82%	112.2%
Perturbed random mixed	736	100%	102.2%	239.1%

shows more obvious advantage when computation domain is divided to less triangular meshes, such as regular quadrilateral grid, random mixed grid, perturbed quadrilateral grid and perturbed random mixed grid. Furthermore, the CPU time comparison demonstrates that faster convergence needs more CPU time except for the case with perfect orthogonal mesh.

As shown in Tab. 4.4, mesh quality has a significant effect on the stability and performance of the solver. Comparing results among regular quadrilateral grid, random mixed grid, perturbed quadrilateral grid and perturbed random mixed grid, quadrilateral mesh shows better convergence. All three solvers use fewest iterations on regular and perturbed quadrilateral mesh. Furthermore, for each mesh type, the skew mesh requires more accurate approximation of PSC as indicated from the comparison between random triangular grid and perturbed random triangular grid. To stabilise the linear systems originated from these skew mesh, SIMPLEC and Semi-coupled solver are better options over SIMPLE, as they show significant improvement when the mesh is perturbed. Tab. 4.5 shows that semi-coupled solver uses more CPU time especially on cases with perturbed grids because of calling twice pressure correction steps for skewness correction.

4.5.3 Sensitivity Validation

In order to validate sensitivity of Semi-coupled implicit solver, the similar approach is applied as used in Chapter 3. Finite difference method, tangent linear method and discrete adjoint method have consistent results, which shows the verified sensitivity obtained by discrete adjoint solver corresponding to Semi-coupled implicit solver. 3D S-bend case with 500k grid case at Re=120 is tested. Objective function is mass averaged pressure loss. Mesh coordinates are control variables. Choosing two random points in computation domain, finite difference results with various step-size are shown in Tab. 4.6. Tangent linear sensitivity accumulating at

corresponding same points are shown in Tab. 4.7. In order to find the best FD result, FD results' absolute errors with tangent linear solution as reference are plotted in Fig. 4.9, and step-size 10^{-5} gives the most accurate result. Discrete adjoint solution gives sensitivity w.r.t. whole control variables, and the corresponding points are picked up for validation comparison. In Tab. 4.7, the absolute error of discrete adjoint sensitivity at two points achieve 10^{-16} or even smaller.

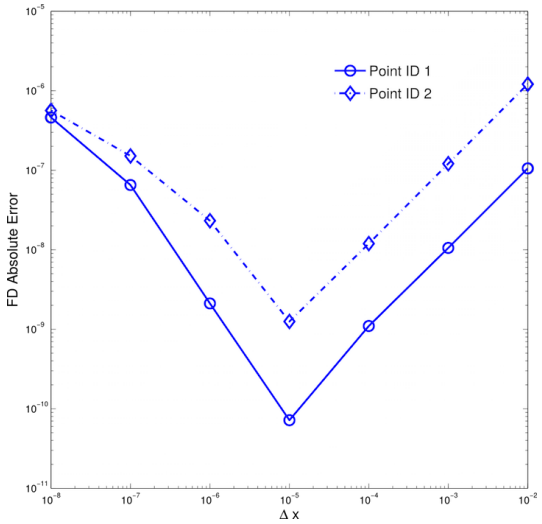


Figure 4.9: Finite difference absolute error of two random points

Table 4.6: Finite difference sensitivity at random two points

Δx	Gradient at Point ID 1	Gradient at Point ID 2
10^{-2}	-3.6370637834792774E-005	2.9226369879342684E-003
10^{-3}	-3.6465511499272907E-005	2.9215488503986364E-003
10^{-4}	-3.6474951947695893E-005	2.9214400409927062E-003
10^{-5}	-3.6475977793770612E-005	2.9214293206791778E-003
10^{-6}	-3.6473934983404984E-005	2.9214048957726100E-003
10^{-7}	-3.6410874315602995E-005	2.9212765539907001E-003
10^{-8}	-3.6015634918804062E-005	2.9208635510229106E-003

Table 4.7: Sensitivity Validation at Random Two Points

Methods	Point ID 1	Point ID 2	$Error_1^{abs}$	$Error_2^{abs}$
FD	-3.6475977793770612E-005	2.9214293206791778E-003	7.172257475836176e-11	1.252928720843483e-09
Tangent	-3.6476049516345371E-005	2.9214280677504569E-003	-	-
Adjoint	-3.6476049516289744E-005	2.9214280677507406E-003	5.562634771208441e-17	2.836272883222080e-16

In Sec. 3.4, as we mentioned above, the standard reverse mode linear system implementation for adjoint variables requires to set $\overline{x} = 0$ before exiting iterative loop. Based on this standard implement, inner tolerance for reverse linear system needs to be set as machine precision in discrete adjoint solver in order to accumulate the accurate sensitivity. However, this implementation is suggested when this linear

system fully converges. In SIMPLE-family algorithms, tolerance of momentum and pressure linear equation only reduces to 1 or 2 magnitudes for inner linear calculation, both of the absolute residuals reducing to 10^{-12} (validation cases) or other user requiring value is carried out via outer iteration non-linear updating of velocity and pressure. For building up discrete adjoint solver for these segregated methods, when we are using fixed-point iteration, the inner linear systems for both primal and adjoint solvers do not need to converge fully. Numerical experience shows that only one iteration for inner linear system of adjoint solver can result in accurate sensitivity. The ratio of CPU time cost by discrete adjoint solver to primal with this improved implementation is 1.08; and using standard implementation with inner tolerance set as 10^{-10} for accurate sensitivity accumulation, the ratio is 2.94.

4.5.4 Non-linear Convergence

Besides 2D test cases, standard SIMPLE, SIMPLEC and the modified Semi-coupled solvers are tested on 3D industrial case. Flow passing through S-bend (shown in Fig. 2.11) channel, which is from Volkswagen Golf[1] are run at Re=600 and Re=1200, which is scaled by inlet height. These cases use 47k hexahedron mesh as shown in Fig.2.12. Same linear solvers are employed for each flow solver. For non-symmetric velocity equation BI-CGSTAB linear solver is applied. CGSTAB solver is called for pressure correction and additional correction equations.

For Re=600 s-bend cases, key parameters work space for tree solvers are conducted firstly. As shown in Fig. 4.10, standard SIMPLE solver has the narrowest work space for pressure under-relaxation with given velocity under-relaxation factor $\alpha_u = 0.9$. Because of the same PSC selection for SIMPLEC and Semi-coupled implicit solvers, these two flow solvers have the similar work space.

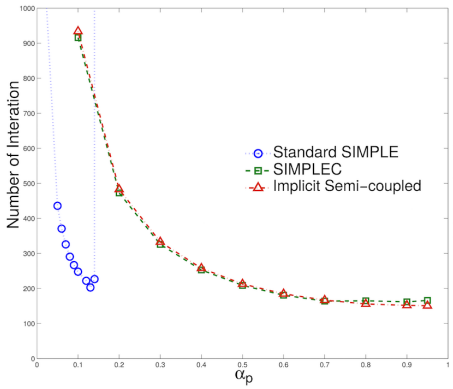


Figure 4.10: Available pressure under-relaxation factor range for standard SIMPLE, SIMPLEC and Semi-coupled Implicit solvers

[1]http://flowhead.sems.qmul.ac.uk

Parameter settings for 3 type solvers' comparison are selected as shown in Tab. 4.8. Inlet uniform velocity and extrapolated outlet velocity are set; non-slip wall is set for channel wall; and based on this fixed mass flow, zero gradient is set for pressure boundary condition. Objective function for sensitivity calculation is mass averaged total pressure loss. All solution values are dimensionless based on Reynold number similarity. Almost the same objective functions are achieved after converging systems to global primal residual with 10^{-10}.

Under this stabilisation, both of primal and discrete adjoint solver show improvement in the convergence behaviour, as illustrated in Fig. 4.12. The convergence of discrete adjoint systems is captured at flow field obtained at same global residual criteria for standard SIMPLE, SIMPLEC and Semi-coupled implicit solver. Among these solvers, Semi-coupled implicit solver damps the convergence oscillation and slightly speeds up the convergence.

Table 4.8: Parameter settings of SIMPLE, SIMPLEC and Semi-Coupled solvers on S-bend cases

Algorithm	α_u	α_p	Scheme	Gradient	Linear solver
SIMPLE	0.9	0.1	Min-Mod	Green Gauss	Bi-CGSTAB/CG
SIMPLEC	0.9	0.9	Min-Mod	Green Gauss	Bi-CGSTAB/CG
SEMI-COUPLED	0.9	0.9	Min-Mod	Green Gauss	Bi-CGSTAB/CG

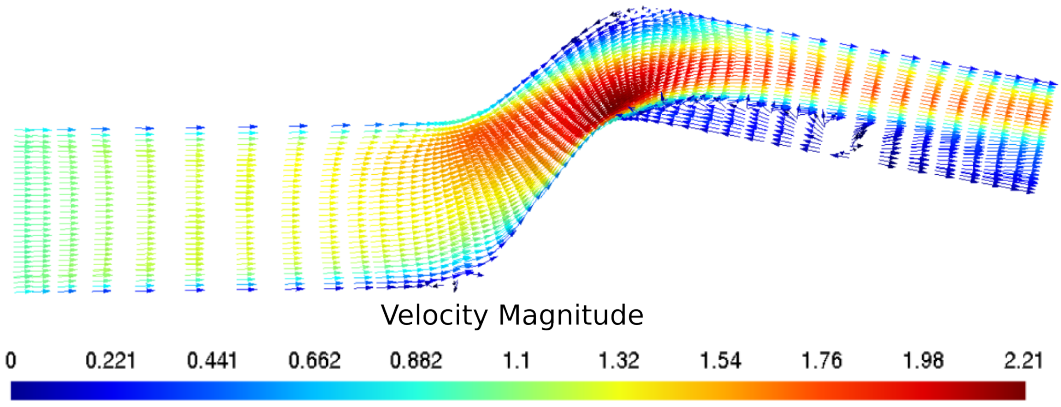
Re=1200 s-bend is carried out for further performance comparison. Through numerical experiments it is found that standard SIMPLE solver fails to converge even when applying 1st order accuracy and extremely small under-relaxation factors for velocity and pressure. In order to stabilise the system as much as possible at this Reynold number, smaller pressure under-relaxation factors and 1st order upwind scheme (listed in Tab.4.9) are applied for Semi-coupled implicit solver. Velocity fields

Table 4.9: Parameter settings of semi-coupled solvers

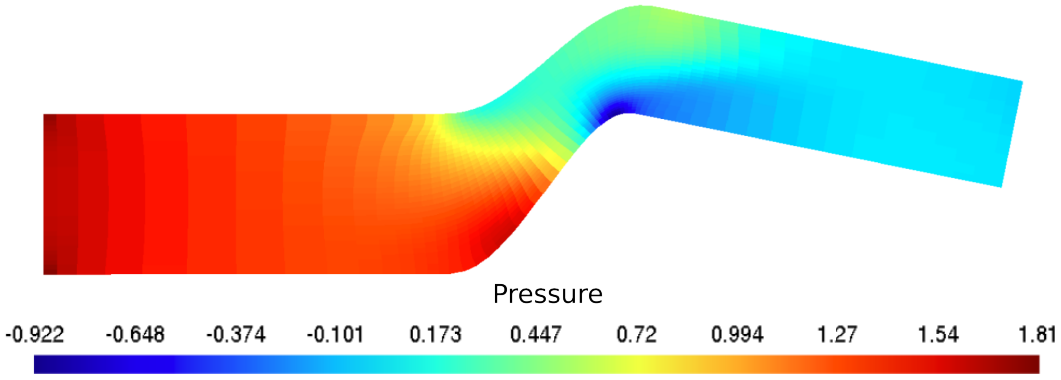
Algorithm	α_u	α_p	Scheme	Gradient	Linear solver
SEMI-COUPLED	0.7	0.4	1st UPS	Green Gauss	Bi-CGSTAB/CG

from four views are demonstrated in Fig.4.13. Since the internal cut-plane velocity and pressure solution is illustrated in Fig.4.14, flow details are captured: flow become complex and generate bubble when passing through the bending area, and bubble reattaches before outlet.

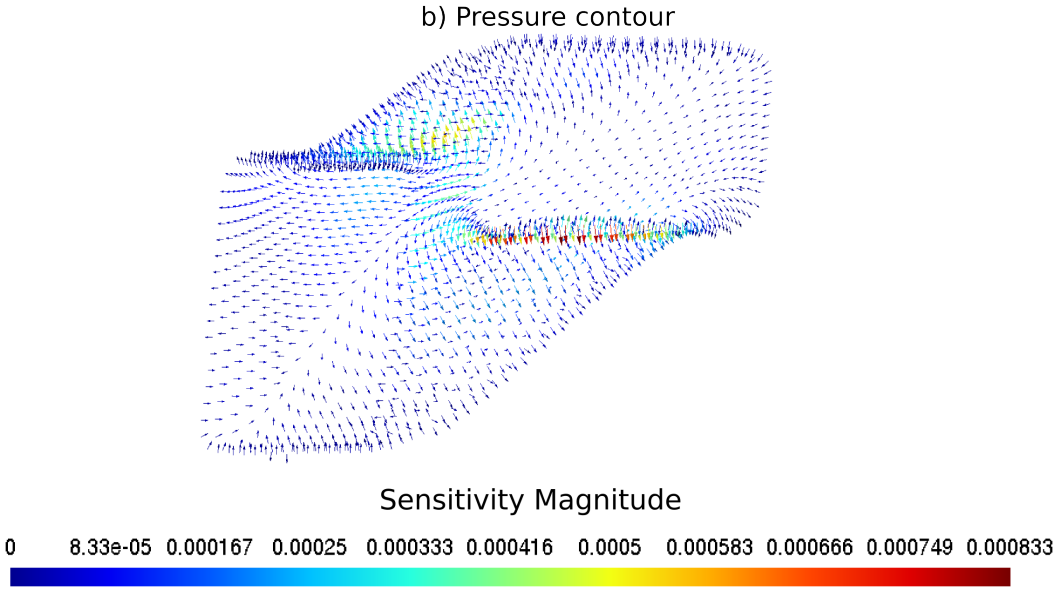
In order to challenge solvers further, Re increases as 6000 for this coarse mesh. At this Reynolds number, the flow shows turbulence properties physically. From mathematical point of view, turbulence model introduces numerical viscosity to stabilize the non-linear system. Normally laminar solver is very difficult to converge the



a) Non-dimensional velocity vector field



b) Pressure contour



c) Surface normal sensitivity

Figure 4.11: Primal and sensitivity solution at Re=600: internal cut-plane velocity (a), internal cut-plane pressure (b) and bending area surface normal sensitivity (c)

system without turbulence model. However, computation at this Reynolds number can be carried out to test the stabilisation ability of numerical algorithms. The comparison among 3 solvers is set as shown in Tab. 4.10. Fig. 4.16 illustrates the primal (left) and adjoint (right) convergence comparisons. As it is shown that standard

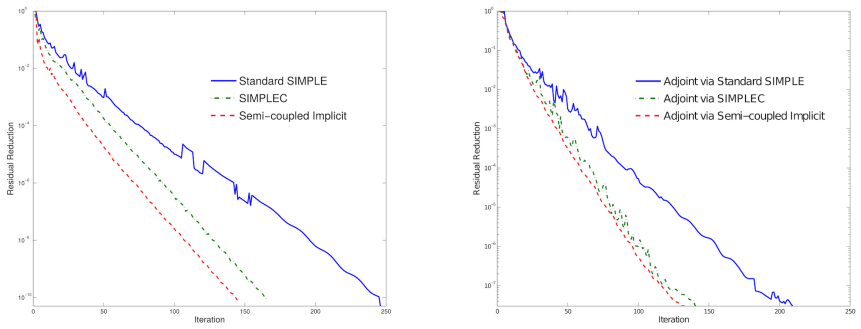


Figure 4.12: Convergence histories for a S-bend test case: standard SIMPLE Solver vs. Semi-coupled Implicit Solver at Re=600

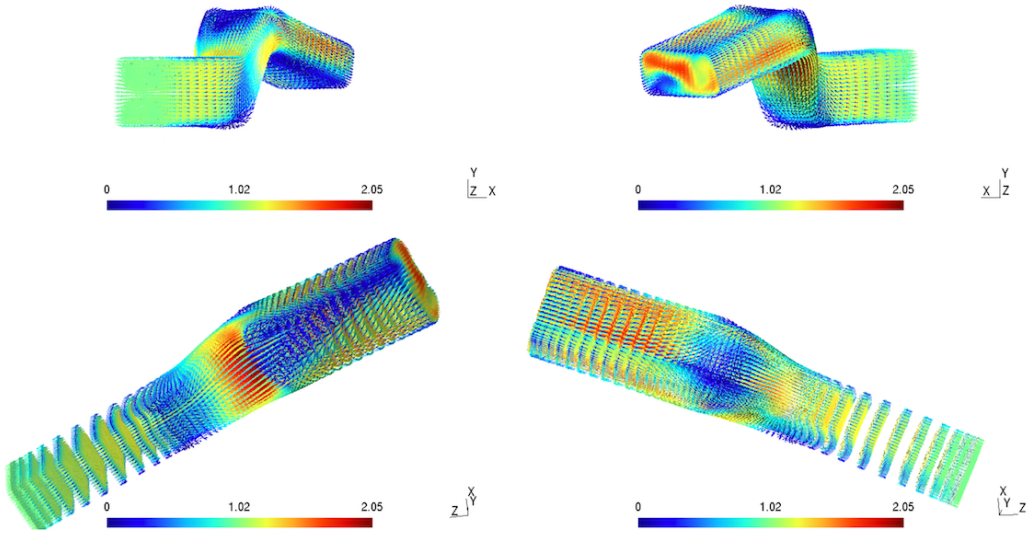


Figure 4.13: Semi-coupled Implicit Solver Velocity Solution at Re=1200: inlet view (up left), outlet view (up right), bottom view (down left) and top view (down right)

SIMPLE solver only converges flow to LCO with residual rounding 10^{-2}, SIMPLEC and Semi-coupled implicit solver converge fully to residual 10^{-11}. Based on the same outer iteration solution, all the three solvers can achieve large adjoint residual reduction (to 10^{-10}), but adjoint via standard SIMPLE solver shows obvious oscillation. And the sensitivity solution is not reliable because the adjoint solution is based on a random choice from the changing flow fields rather than stable one.

Table 4.10: Parameter settings for solver comparison at Re=6000

Algorithm	α_u	α_p	Linear solver	CPU Time	Continuity Residual
SIMPLE	0.7	0.1	Bi-CGSTAB/CG	LCO	10^{-2}
SIMPLEC	0.7	0.1	Bi-CGSTAB/CG	100%	10^{-11}
SEMI-COUPLED	0.7	0.1	Bi-CGSTAB/CG	155%	10^{-11}

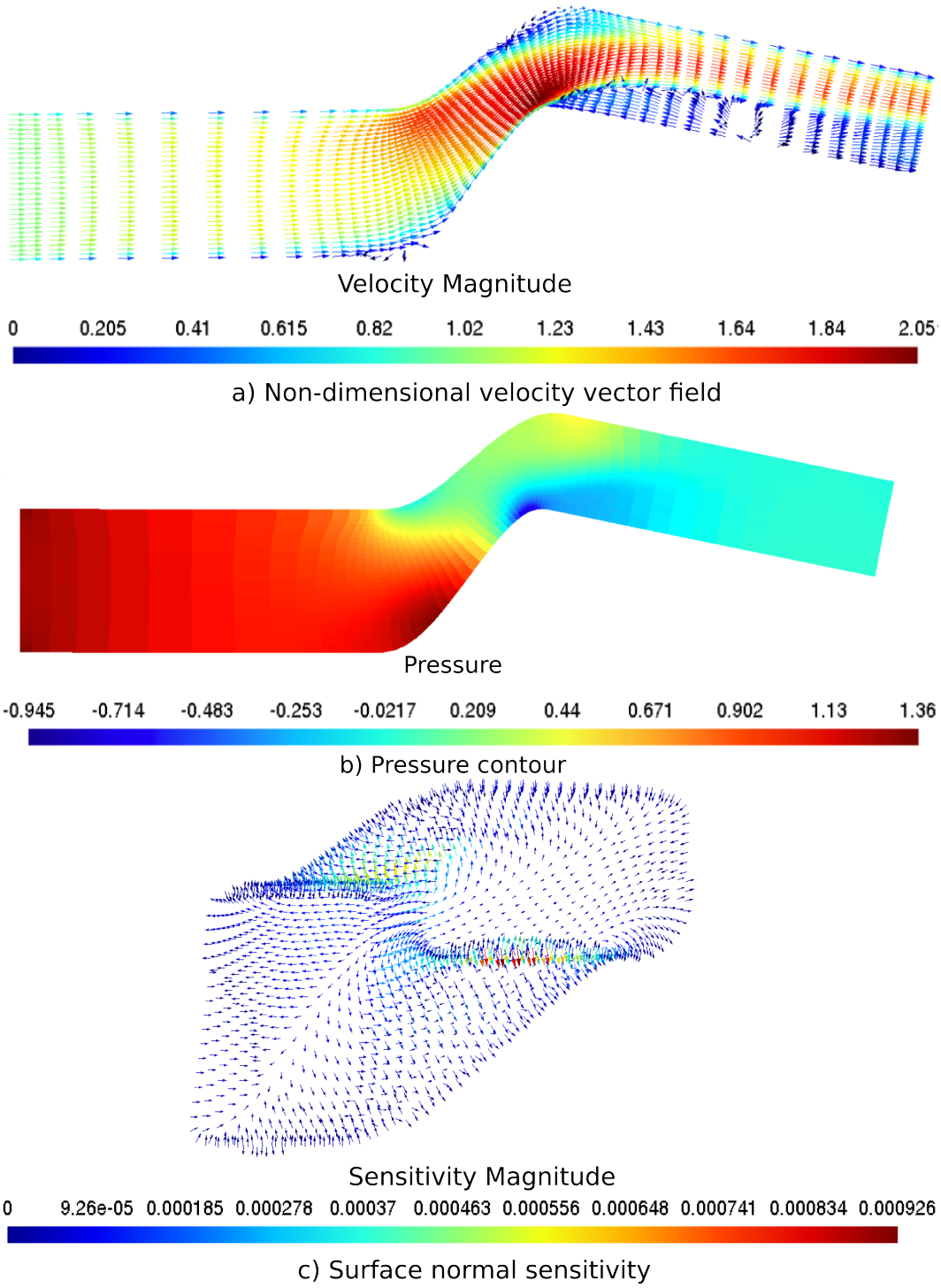


Figure 4.14: Semi-coupled implicit solver solution at Re=1200: internal cut-plane velocity (a), internal cut-plane pressure (b) and bending area surface normal sensitivity (c)

It should be noticed that at this coarse mesh, algebraic multi-grid is not reliable to use for compute pressure correction equation even it can converge the linear system at each outer iteration as seen in Appendix B.3, because numerical experiments shows that non-linear system might diverge when applying algebraic multi-grid solver for

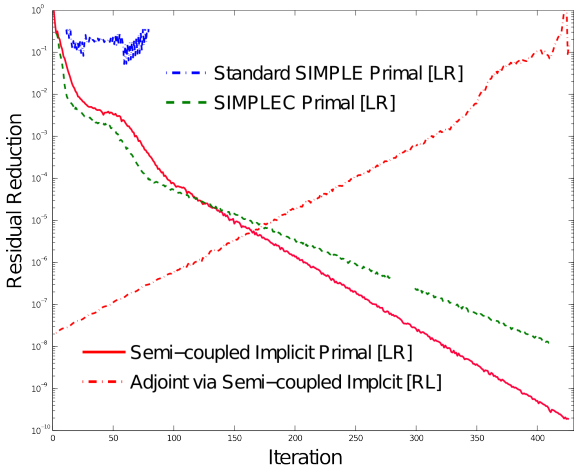


Figure 4.15: Semi-coupled implicit solver solution at Re=1200: primal and adjoint convergence

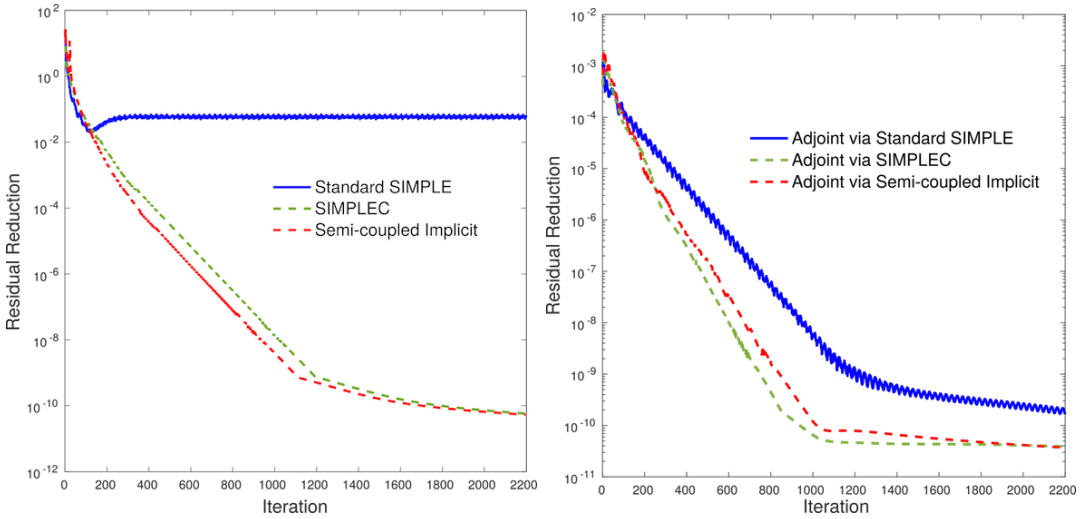


Figure 4.16: Convergence histories for 47k S-bend test case: standard SIMPLE Solver vs. Semi-coupled Implicit Solver at Re=6000

pressure correction equation.

Fine grid test case

AMG technique plays important role when cases running on fine grid. As discussed in Sec. 4.4, AMG method is widely used for solving pressure Poisson equation for the fine grid cases. BoomerAMG solver and PCG with BoomerAMG preconditioned solvers are compared to show performance with variation of coarsen types and number of cycle levels in Appendix B.3. Poisson systems are obtained from 3 dimensional S-bend test cases with increasing grid numbers from 47k, 130k and 500k. In order to make reasonable comparison, linear systems are obtained at system converge to residuals with the same order of magnitude. For fixed number of multi level, BoomerAMG solver does not perform well with 2 level and 5 level cycles. All the test

cases diverge. BoomerAMG with 10 levels converge majority cases expect cases with 47k grid using RS and Falgout coarsen types. In contrast, BoomerAMG preconditoned PCG solver shows more robust behaviour as no case diverges. Solver with fixed number of level cycles takes more iteration number and CPU time compared with adaptive multi level cycles. Among all test coarsen types, Hybrid Modified Independent Set (HMIS) algorithm shows most stable performance on pressure correction linear convergence.

At Re=120, 500k grid cases are tested via standard SIMPLE, SIMPLEC and Semi-coupled Implicit solvers with parameters shown in Tab. 4.11.

Table 4.11: Parameter settings for solver comparison on 500k grid

Algorithm	α_u	α_p	Linear solver	CPU Time	Continuity Residual
SIMPLE	0.9	0.1	Bi-CGSTAB/CG	-(Divergence)	-
SIMPLEC	0.9	0.1	Bi-CGSTAB/CG	100%	10^{-11}
SEMI-COUPLED	0.9	0.1	Bi-CGSTAB/AMG	45.8%	10^{-13}

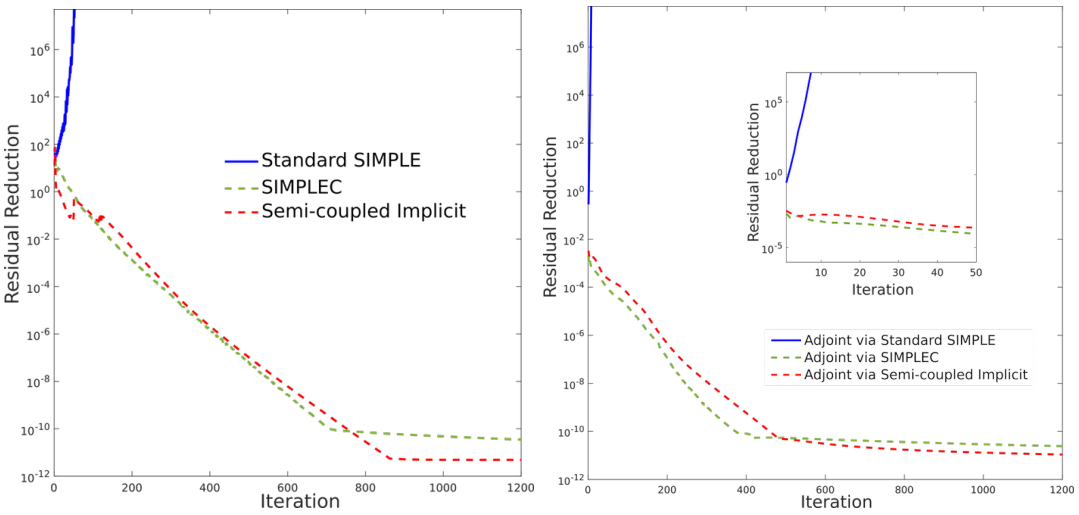


Figure 4.17: Convergence histories for 500k S-bend test case: standard SIMPLE Solver vs. Semi-coupled Implicit Solver at Re=120

In this chapter, SIMPLE-family algorithms are discussed from two different angles, idea of "prediction-correction" and pressure Schur complement. Standard SIMPLE and SIMPLEC algorithms are introduced based on different PSC. New PSC for SIMPLE-family algorithms are proposed with detail implementation. Performance of convergence shows that SAI and SIMPLEC inspired PSC can work with larger range of pressure under-relaxation compared with standard SIMPLE; and PSC based on "neighbour-correction" gives the largest work space. Semi-coupled implicit coupling is proposed to stabilise the flow computation and according PSC is derived for pressure correction equation. For skew grid, IDC scheme is implemented in GPDE with the scheme level enhancement for robustness; 2 step pressure correction strategy is

also introduced for skewness correction. Moreover, AMG solver is linked to solve pressure Poisson equation as the stabilisation effect of the linear solver. With a comprehensive effect from methods mentioned above, the flow solver shows improved convergence with larger work space and works on larger range of cases as discussed. Based on the primal stabilisation improvement, discrete adjoint solver with improved implementation shows stable and more efficient performance. These improvement let flow and sensitivity solvers work well to offer robust performance and accurate gradient during the optimisation, which results in an effective and efficient optimisation loop.

Chapter 5

Shape optimisation case study

Shape optimisation is the major application for adjoint sensitivity computation. Gradient-based optimisation loop is carried out in general as:

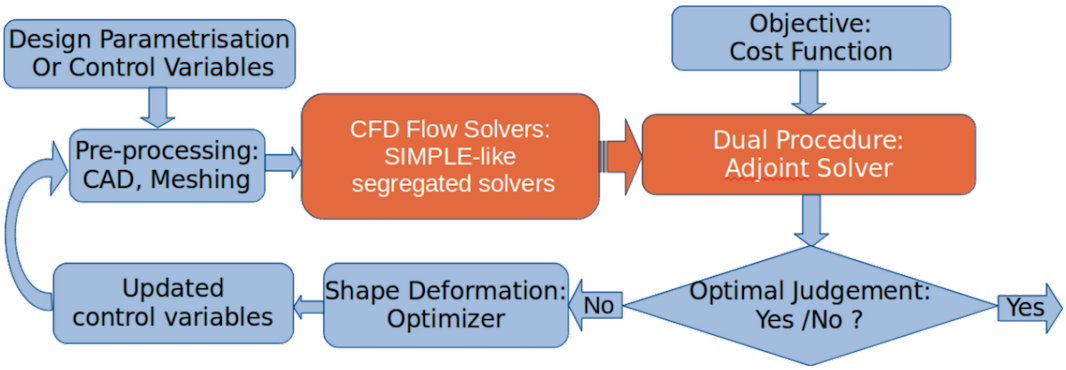


Figure 5.1: Shape optimisation loop based on discrete adjoint method

As it is illustrated in Fig. 5.1, the gradient based design loop can be divided into several steps. At the beginning, all the design parameters or control variables are targeted and labelled in the CAD or mesh geometry. Then, the flow field is solved numerically by the stabilized CFD solver and used to get the sensitivity of the pre-defined cost function using discrete adjoint solver. Through mesh deformation with the sensitivity, geometry of the shape is altered optimally and the controlled variables are updated accordingly. Therefore, a gradient-based optimisation loop using discrete adjoint is built to search the optimum shape. Until the objective function reaches the desired value, the optimisation loop terminates. As we can see here, the gradient-based numerical shape optimisation can be carried out automatically until the final shape satisfy the user defined criteria. In industrial shape optimisation case, the most expensive parts are flow fields computation and objective surface sensitivity. In order to reduce the whole time, the key part is to develop efficient flow solver and discrete adjoint solver. Besides, the flow solver and adjoint solver are required to perform more stable and robust compared with single flow computation case, because the mesh iterates as the shape changes and due to that the mesh quality may

not maintain. In this chapter, gradient-based optimisation strategy is introduced in Sec. 5.1 and back-tracking line search method is carried out for industrial cased. Node-based and CAD-based approaches for selection of geometric design variables are introduced in Sec. 5.2, and CAD-based optimisation using in-house tool NsPCC is applied for the case study. In Sec. 5.3, mesh deformation is discussed based on the linear elasticity analogy method. S-bend air duct optimisation cases at two Reynolds numbers are demonstrated in Sec. 5.4.

5.1 Gradient-based optimisation strategy

At ith iteration of optimisation loop, the next step control variable $\boldsymbol{\alpha_{i+1}}$ is updated along a search direction $\mathbf{p}_i$ as:

$$\boldsymbol{\alpha_{i+1}} = \boldsymbol{\alpha_i} + w_i \mathbf{p}_i \tag{5.1}$$

where the positive scalar w_i is the step length. In order to guarantee that the objective function L can be reduced along the search direction, normally line search algorithms require $\mathbf{p}_i$ to satisfy:

$$\mathbf{p}_i^T \nabla L_i < 0 \tag{5.2}$$

which leads $\mathbf{p}_i$ to be a descent direction. In the steepest descent method $\mathbf{p}_i = -\nabla L_i$. There is still another task for optimisation algorithm here that is to find a proper value of w_i, which reduces objective function from $L(\boldsymbol{\alpha})$ to $L(\boldsymbol{\alpha} + w\mathbf{p})$

For gradient-based optimisation, backtracking line search algorithm is used to drive the iteration loop, which is based on Armijo-Goldstein condition to confine the step length w_i to guarantee a decrease of objective function[136]. In backtracking line search, the decent direction can be chosen as unit normalised gradient negative direction:

$$\mathbf{p}_i = -\frac{\nabla L_i}{\|\nabla L_i\|} \tag{5.3}$$

The algorithm of choosing step length w is implemented as:

1. Choose initial step length $w_0 > 0$ and two control parameters $\rho, c \in (0, 1)$
2. Set $t = -c\mathbf{p}^T \nabla L$ and internal iteration counter $j = 0$;
3. Calculate $\Delta L = L(\boldsymbol{\alpha}) - L(\boldsymbol{\alpha} + w_j \mathbf{p})$;
4. Judge: if Armijo-Goldstein condition is satisfied:

$$\Delta L \geqslant w_j t, \tag{5.4}$$

return $w = w_j$ as the current step length; if not, let $w_{j+1} = \rho w_j$ and repeat step 3.

Therefore, for each optimisation iteration the step length w is determined in order to decrease the objective function sufficiently. And entire backtracking line search algorithm for shape optimisation can be expressed as follows:

1. Set iteration counter $i = 1$, and call flow solver and discrete adjoint solver to calculate temporary objective function L_{tmp} and gradient ∇L_{tmp};
2. Compute a decent direction $\mathbf{p}_i$;
3. Choose step length w_i as mentioned above, update control variable as $\boldsymbol{\alpha_{i+1}} = \boldsymbol{\alpha_i} + w_i \mathbf{p}_i$;
4. Compute current step objective function L_i and gradient ∇L_i;
5. Judge: if $L \leqslant f$ or $\nabla L \leqslant \epsilon$ where f and ϵ are user defined convergence criteria, terminate with final solution; if not, repeat step 2.

5.2 CAD-based shape optimisation

In CFD shape optimisation, how to choose the parametrisation is a major concern in industrial application [137–139]. There are a number of approaches that include polynomial representation of boundaries, spline representation of boundaries, the design element technique and use of boundary nodes of shape representation [140, 141]. Hojjat etal.[142] pointed out "CAD" (Computer Aided Design) could refer to all the parametrisation methods which a big volume of surface points is represented and controlled by "few" geometrical (CAD) parameters. In that case, when it comes to selection of control variables, shape optimisation can be classified as two major types, node-based and CAD-based approaches.

Node-based shape optimisation is considered as a discrete approach[137], which is based on directly using computational boundary grid nodes' coordinates as control variables. Normally the surface sensitivity is projected along surface normal direction and based on this direction, surface mesh coordinates are perturbed along the gradient direction. In case of aerodynamic design, the described shapes are smooth in general. However, sometimes high-frequency oscillation resulting from 1st order discrete gradient of complex flow cases cannot be appropriately suppressed by the optimiser[22, 143]. In order to prevent oscillation shapes, additional smoothing of the graidents or the displacement is required[144, 145]. Nevertheless, this approach is straight forward to implement and local detail shape changing is captured on the CFD mesh with high density. And it is possible to have a strong local on shape changes

by restricting the changes to a small area[137]. Node-based approach is a valuable tool and applied successfully in sensitivity analysis and design space exproration for shape optimisation[142, 144–147].

Industrial design chains use CAD system to manipulate the geometry [148]. Enclosing CAD description in the optimisation loop is convenient to transfer the optimised shape to CAD model, which is the direct design data for manufacturing. CAD-based optimisation is choosing geometry control points of CAD descriptions as design variables. Therefore, based on chain rule, the adjoint-based sensitivity can be expressed as:

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\alpha}} = \frac{\mathrm{d}L}{\mathrm{d}\mathbf{X}_s}\frac{\mathrm{d}\mathbf{X}_s}{\mathrm{d}\boldsymbol{\alpha}} \tag{5.5}$$

where $\mathrm{d}\boldsymbol{\alpha}$ is CAD parameters and $\mathbf{X}_s$ is the surface node coordinates. The surface of geometry is determined by CAD-based parametrisations, which means the shape deformation can be evaluated via a perturbation of design variables. Compared with node-based approach, CAD-based approach requires additional derivative term $\frac{\mathrm{d}\mathbf{X}_s}{\mathrm{d}\boldsymbol{\alpha}}$ in gradient. Finite difference methods have been proposed to offer this CAD part derivative[149, 150]. However, evaluation of these derivatives is expensive and the obtained derivatives are inexact[148]. To overcome these problems, an alternative method that can take advantages from discrete adjoint method for derivative evaluation is proposed by Yu et al. and tested in a case of 2D RAE 2822 transonic aerofoil shape[148]. A set of Non-uniform Rational B-splines (NURBS) surface patches[151] are used for geometry shape description. The boundary representation (BRep) is writen in the STEP standard[152, 153].

In this chapter, CAD-based parametrisation termed 'NURBS-based parametrisation with continuity constraints' or NsPCC method[22] is applied for S-bend air duct optimisation case study. A NURBS is in general defined[151] as:

$$X_s(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,j}(u,v)P_{i,j} \tag{5.6}$$

where $P_{i,j}$ are control points, and u and v are the independent parametric variables of surface mesh point. $B_{i,j}(u,v)$ is the rational basis function. In-house CAD tool NsPCC, using AD to obtained the derivatives, works with following five functions:

1. Load STEP file: reading input STEP file and obtaining the surface and curves information;

2. Find parameters: matching the surface nodes and the CAD geometry;

3. Calculate derivatives: calculating derivative $\frac{\mathrm{d}\mathbf{X}_s}{\mathrm{d}\boldsymbol{\alpha}}$;

4. Perturb the geometry: with the updated control variables $\boldsymbol{\alpha}$, the perturbed

new B-Splines are generated;

5. Write STEP: write the perturbed geometry in to new STEP file.

Meanwhile the updated surface nodes data is also provided by NsPCC, which is matched with the updated STEP file.

5.3 Mesh Deformation Algorithm

At each optimisation iteration, updating meshes are required by the flow solver and adjoint solver for new objective function and sensitivity computation. Since both of the updated STEP file and new surface nodes are obtained via NsPCC as discussed above, the new mesh can be obtained via re-meshing and mesh deformation approaches. However re-meshing the updated geometry normally can not be completed automatically, which leads interruption of optimisation iteration loop and meanwhile it takes long time to generate a new mesh based on the updated geometry. Rather than re-meshing approach, mesh deformation method directly deforms the current mesh nodes with given conditions to calculate the new mesh nodes' position, which can be implemented numerically and inserted into optimisation loop easily. Therefore, mesh deformation strategy is considered in order to propagate the perturbed surface displacement through the whole computation domain to rebuild the mesh. In shape deformation case, the model of the entire domain is considered as a homogeneous elasticity body. The linear elasticity analogy is applied to govern the mesh deformation[154][155]. For equilibrium state, the body force is zero (when boundary displacement is given), the simplified equation can be obtained[156]

$$\nabla \cdot \sigma = 0 \quad in \quad \Omega \tag{5.7}$$

where σ is stress tensor. Based on isotropic material assumption, stress tensor is given by

$$\sigma = 2\mu\epsilon + \lambda Tr(\epsilon)\mathbf{I} \tag{5.8}$$

where Tr is the trace. λ and μ are material properties of elastic material, which are related to Young's modulus E and Poisson's ratio ν:

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}, \quad \mu = \frac{E}{2(1+\nu)}. \tag{5.9}$$

ϵ is the strain tensor

$$\epsilon = \frac{1}{2}(\nabla\delta\mathbf{X} + \nabla\delta\mathbf{X}^T) \tag{5.10}$$

where $\delta\mathbf{X}$ is the displacement vector.

Under the FVM framework and and applying Gauss's theorem for solving mesh perturbation equation, Eq. (5.7) can be expressed as

$$\oint_{\partial\Omega}\left(\frac{E}{2(1+\nu)}(\nabla\delta\mathbf{X}+\nabla\delta\mathbf{X}^T)\cdot\mathbf{n}+\frac{\nu E}{(1+\nu)(1-2\nu)}\nabla\cdot\delta\mathbf{X}\mathbf{n}\right)\mathrm{d}S=0 \tag{5.11}$$

where the mesh deformation relies on the values selection of Young's modulus and Poisson's ratio. There are many successful cases of implementation for mesh perturbation proposed[21, 157, 158]. Biedron et al.[158] introduced a simple and robust approach, which is implemented in GPDE. The Poisson ratio ν is given a uniform value of zero. And then the weak formulation of nodes' displacement can be simplified as:

$$\oint_{\partial\Omega}(\frac{E}{2(1+\nu)}\nabla\delta\mathbf{X}+\nabla\delta\mathbf{X}^T)\cdot\mathbf{n}\mathrm{d}S=0 \tag{5.12}$$

and Dirichlet boundary with surface displacement $\mathbf{X}_s$ is imposed on Eq. (5.12). The discretilized equation can be written as:

$$\mathbf{K}\delta\mathbf{X}=f(\delta\mathbf{X}_s) \tag{5.13}$$

where $\mathbf{K}$ is the coefficient stiffness matrix and $f(\delta\mathbf{X}_s)$ is the right hand side function dependent on the given surface displacement. According to [21, 158], Young's modulus E is taken as inversely proportional to the distance from the nearest solid boundary (wall distance) or its higher order. In that case the coefficient can be achieved:

$$k_{ij}=\sum_{N_F}(-\frac{A}{d_w^n}\frac{\Delta S}{l_{IJ}}) \tag{5.14}$$

where A and n is user defined constant, for our computation, the default setting is $A=1$ and $n=3$. d_w wall distance is approximated based on method [159]. Firstly, wall distance variable ϕ is governed by a transport equation:

$$\int_{\partial\Omega}\nabla\phi\cdot\mathrm{d}\mathbf{S}=-\int_{\Omega}\mathrm{d}\Omega \tag{5.15}$$

with Dirichlet boundary condition $\phi=0$ at walls and Neumann boundary condition $\frac{\phi}{\partial n}=0$ at other boundaries. From the solution of wall distance variables, wall distance can be obtained as:

$$d_w=\sqrt{\nabla\phi\cdot\nabla\phi+2\phi}-|\nabla\phi|. \tag{5.16}$$

Because of cell-centred data structure applied in GPDE, the deformed mesh displacement is located at cell-centroids for each cell. Extra two mapping are used. The boundary coordinates of grid node can be transformed to boundary face center (vertex to element mapping); cell-centroids solutions are mapped to nodes coordinates

that construct the deformed new mesh.

5.4 S-bend air duct case study

A three dimensional S-bend air duct case is shown in Fig. 2.11. Two shape optimisation cases with different Reynolds numbers are carried out to show the shape changing according to respective flow physics. The objective function, mass average pressure drop, is defined as:

$$L = \frac{\int_{in} P_{in}\mathbf{u}\cdot\mathbf{n}\mathrm{d}S}{\int_{in}\mathbf{u}\cdot\mathbf{n}\mathrm{d}S} - \frac{\int_{out} P_{out}\mathbf{u}\mathbf{n}\cdot\mathrm{d}S}{\int_{out}\mathbf{u}\cdot\mathbf{n}\mathrm{d}S}. \tag{5.17}$$

Firstly, the low Reynolds number (Re=60) S-bend CAD-based optimisation is developed and the results are shown in Fig. 5.2. The results verify the gradient-based optimization loop that the objective function, pressure drop, continuously reduces with the evolution of iterations. After 103 optimisation iterations, both of objective function and gradient values converge as shown in Fig. 5.2, where the normalised gradient is the two-norm of the gradient vector.

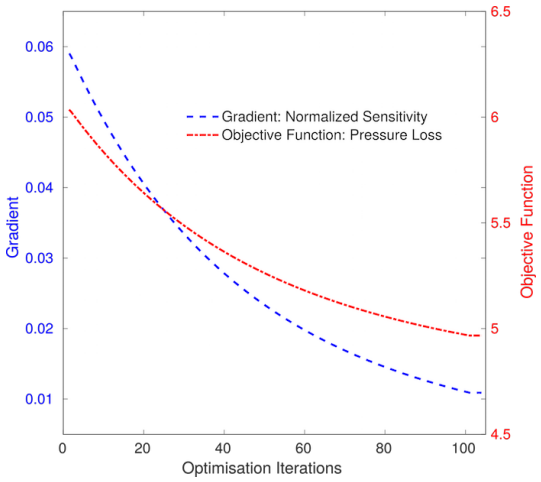


Figure 5.2: Objective function and gradient convergence history at Re=60

The initial shape is modified to an optimised shape as shown in Fig. 5.3. According to mass continuity:

$$\rho_1 u_1 A_1 = \rho_2 u_2 A_2 \tag{5.18}$$

where A_1 and A_2 are two cross-section areas along the duct; ρ_1, ρ_2, u_1 and u_2 are the mean density and velocity at respective positions. At Re=60, flow in air duct is viscosity dominant. Flow in the duct is laminar flow. Therefore, pressure loss in this

steady incompressible flow case can be given by the relation [160]:

$$\frac{1}{\gamma}\frac{\Delta p}{\Delta x} = \lambda \frac{l}{D}\frac{u^2}{2g} \tag{5.19}$$

where γ is the specific weight of fluid; Δx is the elementary length; λ is the coefficient of resistance; D is the pipe diameter; u is the mean velocity in the cross section and g is the acceleration of gravity. The entire duct shape modification is shown in Fig. 5.3. As shown in Fig. 5.4, the optimised shape of detail cross-section illustrates the increasing area of the bending area compared with the initial cross section shape. Fig. 5.6 illustrates the initial and optimised streamlines, where we can find that the secondary flow motion is not strong at this low Reynolds number. The pressure loss is mainly due to the skin friction. Based on Eq. (5.18), when cross section area increases, the velocity reduces. Meanwhile the equivalent diameter D increases because of the enlarged cross section area. Therefore, the enlarged bending cross section leads to pressure loss reduction according to Eq. (5.19), where the constant coefficient of resistance is assumed due to the smooth transition segment of the channel. Same outlet pressure is set as reference value. The pressure distributions along the air duct are demonstrated in Fig. 5.5, and smaller mean pressure values in optimised duct compared with initial ones leads less pressure drop in total. For this low Reynolds number shape optimisation, standard SIMPLE algorithm keeps stable performance during all iterations with the initial parameter settings.

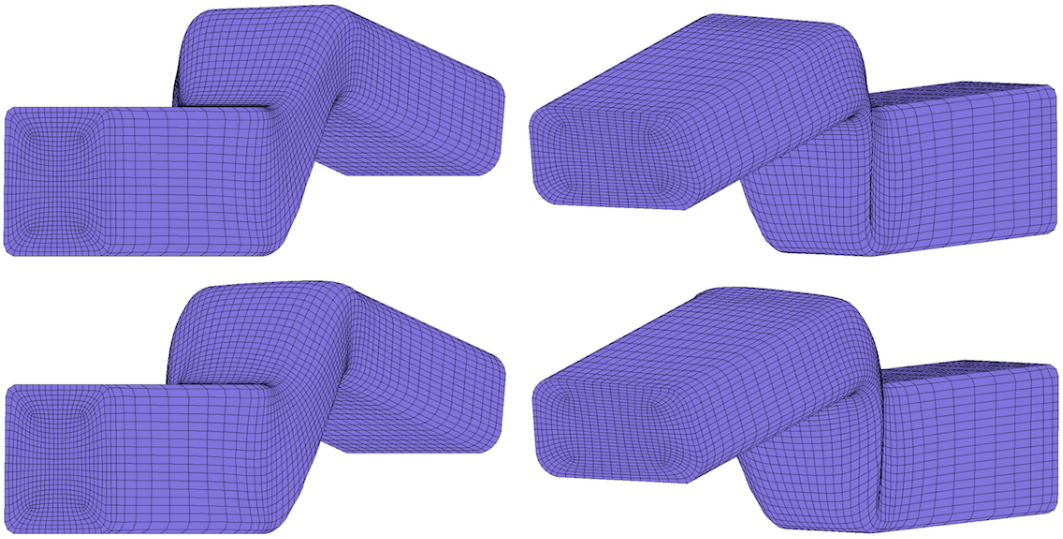


Figure 5.3: Shape optimisation of S-bend air duct at Re=60: initial shapes (upper) and optimised shapes(lower)

When Reynolds number increases, flow becomes complex and secondary currents happens resulting from the curvature. The secondary flow in the bent duct is known as Dean vortices[161, 162]. Flow in a curved rigid duct is characterized by two non-

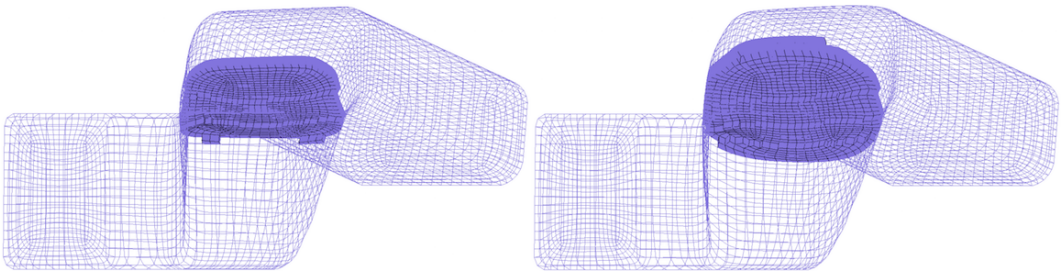


Figure 5.4: Initial cross-section shape (left) and optimised cross-section shape (right) at Re=60

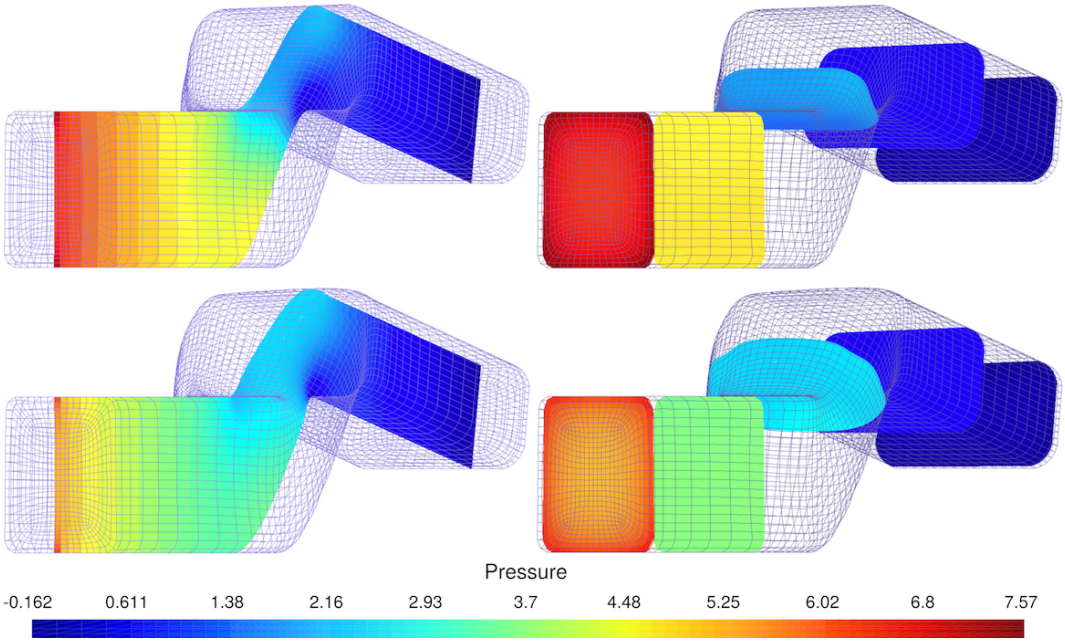


Figure 5.5: Pressure fields distribution at Re=60: initial pressure distribution (upper) and optimised pressure distribution (lower)

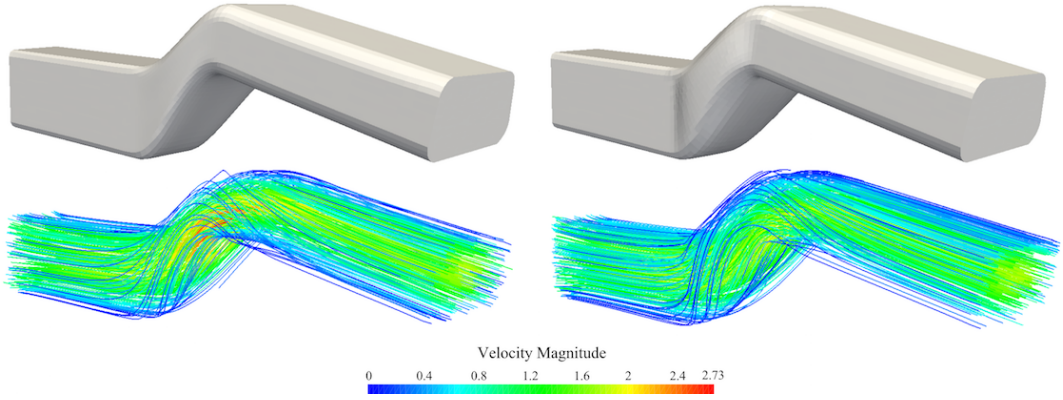


Figure 5.6: Streamlines of the initial flow (left) and the optimised flow (right) at Re=60

dimentional parameters, Dean number D_n and the curvature ratio σ[162]. The Dean number is defined as:

$$D_n = \sigma^{1/2} Re = \left(\frac{r}{R}\right)^{\frac{1}{2}} Re \tag{5.20}$$

where r is the radius of circle or the equivalent radius and R is the radius of curvature of the duct path. For a loosely coiled pipe, $R >> r$ and $\sigma << 1$, D_n plays the role of a "Reynolds number" of the flow and leads to so called Dean-number similarity[162, 163], which is used to show the influence of the secondary flow motion as shown in Fig. 5.7. Due to the secondary flow motion, the local loss increases. In order to decrease the pressure loss, it is necessary to consider reducing the secondary vortices effects.

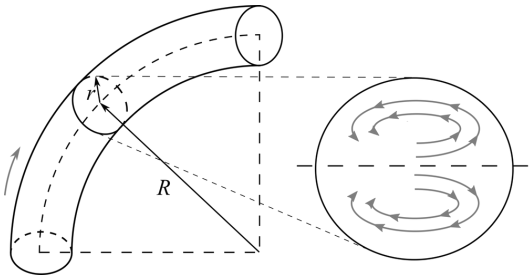


Figure 5.7: Schematic diagram of Dean vortices in curvature duct

Optimisation case at Re=300 demonstrates the shape changing (shown in Fig. 5.8) based on this flow physics. In Fig. 5.9, the optimised shape shows more detailed changing around the bending cross section. Compared with the optimised shape at lower Reynolds, optimised shape at Re=300 shows enlarged cross-section area in general but also the squeezed-in sides, where this detailed change suppresses the Dean vortices in order to reduce the energy loss around the bent segment. Fig. 5.11 illustrates that streamlines become smooth because of the suppression effects and less swirl flow arises in optimised flow channel. As shown in the velocity magnitude field and streamlines, the secondary flows are weakened after the optimisation. The velocity uniformity is also improved as seen the reduced difference between the highest and lowest velocities at the same cross-section.

The similar optimisation study demonstrates the suppression effects of Dean vortices thanks to this "strake-like" shape[22], but more accurate solution is achieved from lower residual convergence of both primal and adjoint solvers. The robust and stable performance of flow and sensitivity solvers based on improved semi-coupled implicit solver are applied here since standard SIMPLE algorithm fails to converge the flow during the optimisation iterations due to the shape changing. Converged objective function and gradient are obtained after 120 optimisation iterations as shown in Fig. 5.12.

The adjoint sensitivity optimisation design loop is completed with iterative strategy, selection of control variables and mesh perturbation method. According to the gradient information, the control variables are updated based on backtracking line search method in order to guaranteed sufficient objective reduction with optimal step length. NsPCC methodology brings CAD description into the optimisation

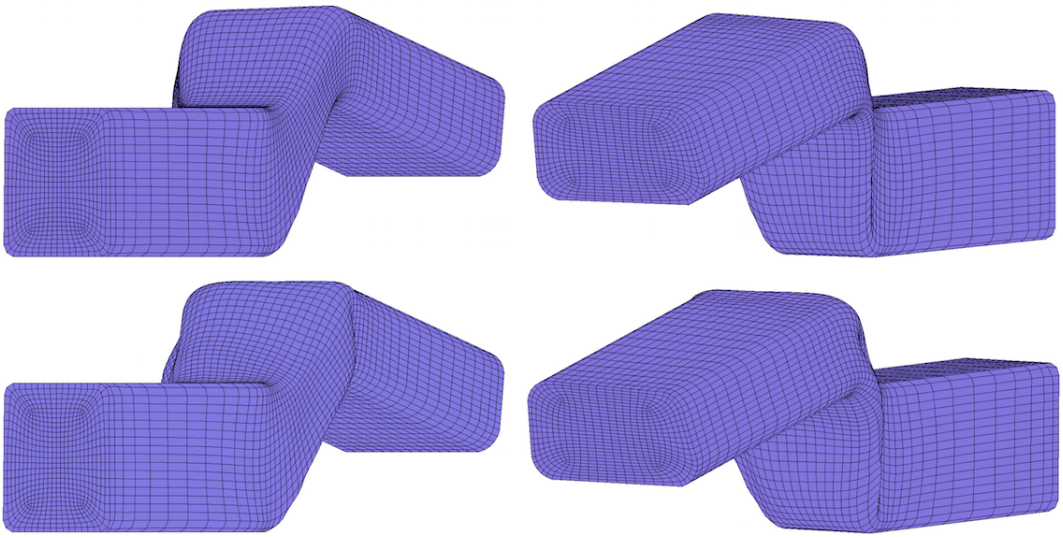


Figure 5.8: Shape optimisation of S-bend air duct at Re=300: initial shapes (upper) and optimised shapes(lower)

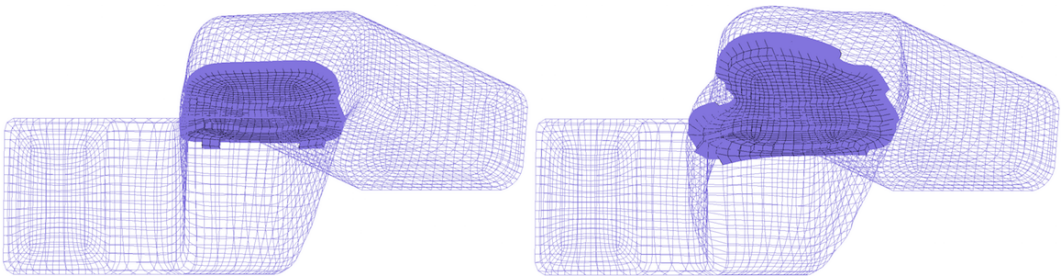


Figure 5.9: Initial cross-section shape (left) and optimised cross-section shape (right) at Re=300

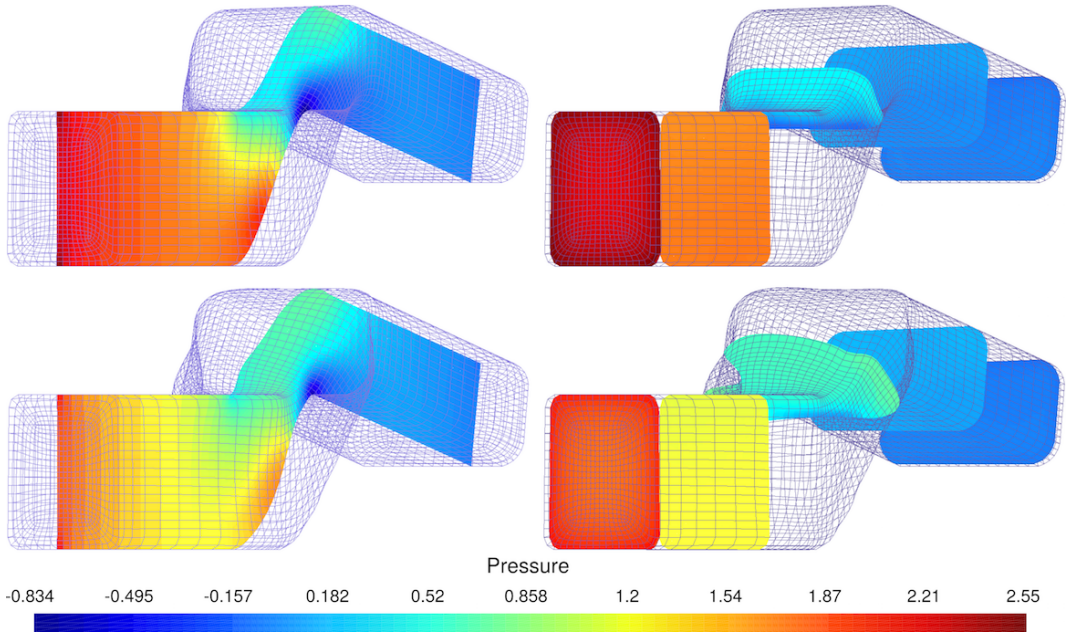


Figure 5.10: Pressure fields distribution at Re=300: initial pressure distribution (upper) and optimised pressure distribution (lower)

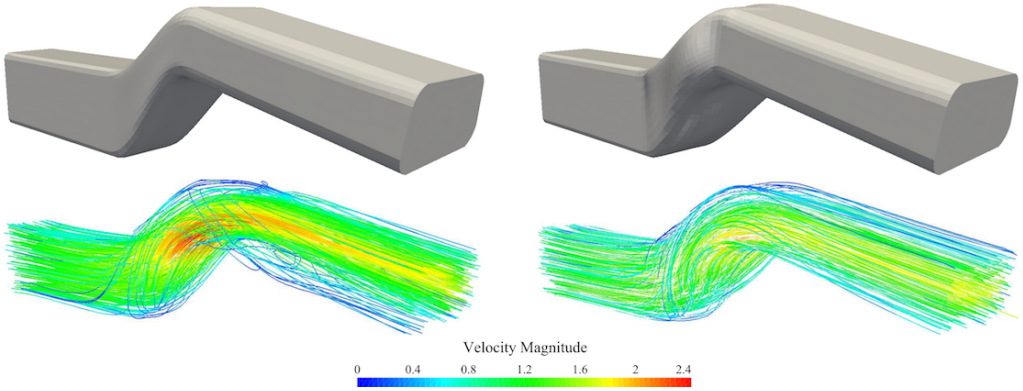


Figure 5.11: Streamlines of the initial flow (left) and the optimised flow (right) at Re=300

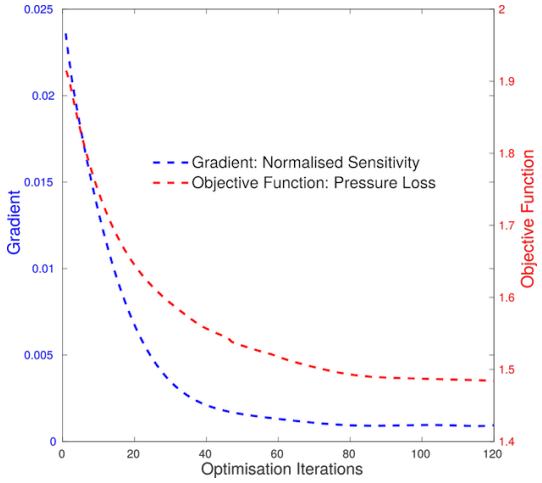


Figure 5.12: Objective function and gradient convergence history at Re=300

loop. NURBS control points are selected as design variables. With the continues formulation of BRep, the modification of movable surface is perturbed smoothly. No additional smoothing of the surface displacement is required. Mesh deformation with given boundary displacement is propagated evenly through the entire computational domain governed by linear elasticity mesh deformation methodology. As solution shown in Fig. 5.3 and Fig. 5.8, not only the average flow sensitivity property can be evaluated but also the detail local shape changing tendency can be controlled based on the gradient information. Comparing theses two Reynolds number flows in the air duct, the streamlines show the different flow pattens. Compared with higher Reynolds number case, the lower one has little secondary flow motion as shown in Fig. 5.6. As it is illustrated in Fig. 5.8, optimised shape with higher Reynolds number flow shows strong local detailed deformation due to the Dean vortices.

Numerical experience indicates that standard SIMPLE solver with initial parameter settings can not maintain a stable performance, the flow solver will diverge

after several iterations due to the complex shape and mesh changing. The primal convergence comparison between SIMPLE and semi-coupled implicit solvers at 87th optimisation iteration step is shown in Fig. 5.13. The optimisation loop requires

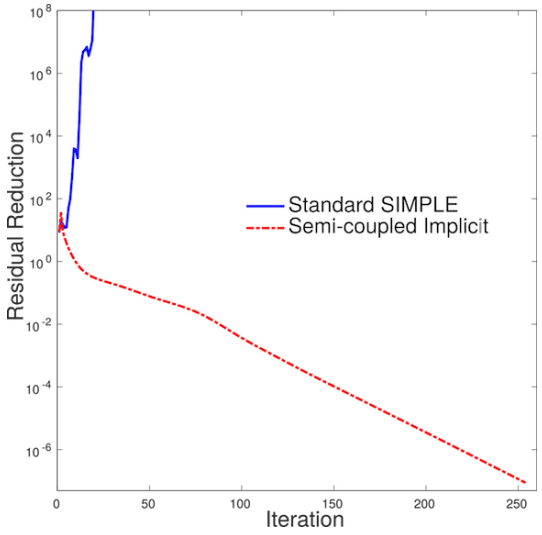


Figure 5.13: Convergence comparison between standard SIMPLE and semi-coupled implicit solver at 87th optimisation iteration step

more robust and stable flow and adjoint solver to achieve the converged solution, where semi-coupled implicit solver combined with multi stabilisation functions can work under wide rang of parameter settings and offer better robust and stable performance during the optimisation loop.

Chapter 6

Case studies in pressure-driven membrane processes

6.1 Geometry of the SWM flow channels

The Spiral Wound Membrane (SWM) is one of the most widely used membrane filters in applications of water treatment and desalination. A common problem faced with water treatments and desalination is the accumulation of the dissolved and suspended solutes onto the membrane surfaces and further into the support layer. Apart from the development of low-fouling membranes, high performance module design also plays a significant role. A schematic diagram of the SWM module is shown in Fig. 6.1 [164]. A membrane envelope has two sheets, glued at the three edges, and the filled permeate channel. The permeate accumulates and flows in a spiral direction toward an inner tube where the permeate is collected. And the feed flows in an axial direction. The spacers are located within the envelopes of the feed channels. Although the SWM has been invented for more than 50 years, the morphology of the commercial module has remained unchanged.

Considering the local detailed flow phenomena, the flow channels with the ladder-type spacer, also called parallel spacer, can be simplified to a two-dimensional flow problem in the transversal direction. Three types of spacer configurations are defined by Schwinge et al. [165], namely zigzag, cavity and submerged. As this thesis focuses on the demonstration of using the adjoint method to develop the sensitivity analysis and the gradient-based optimisation in a spacer-filled RO membrane module, at the early stage, these 2D flow channels with different spacer configurations are considered for simplicity. All the considered flow channels are shown in Fig. 6.2 including open channel in (a), submerged configuration in (b), cavity in (c) and the zigzag in (d).

Six filaments are selected to study the differences between the different configurations [166]. The parameters of these spacer configurations are: channel length

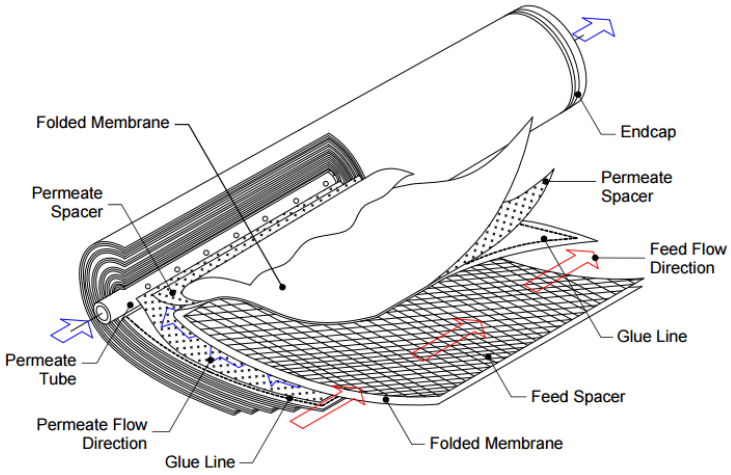


Figure 6.1: A schematic diagram of the SWM module used in water treatments and desalination

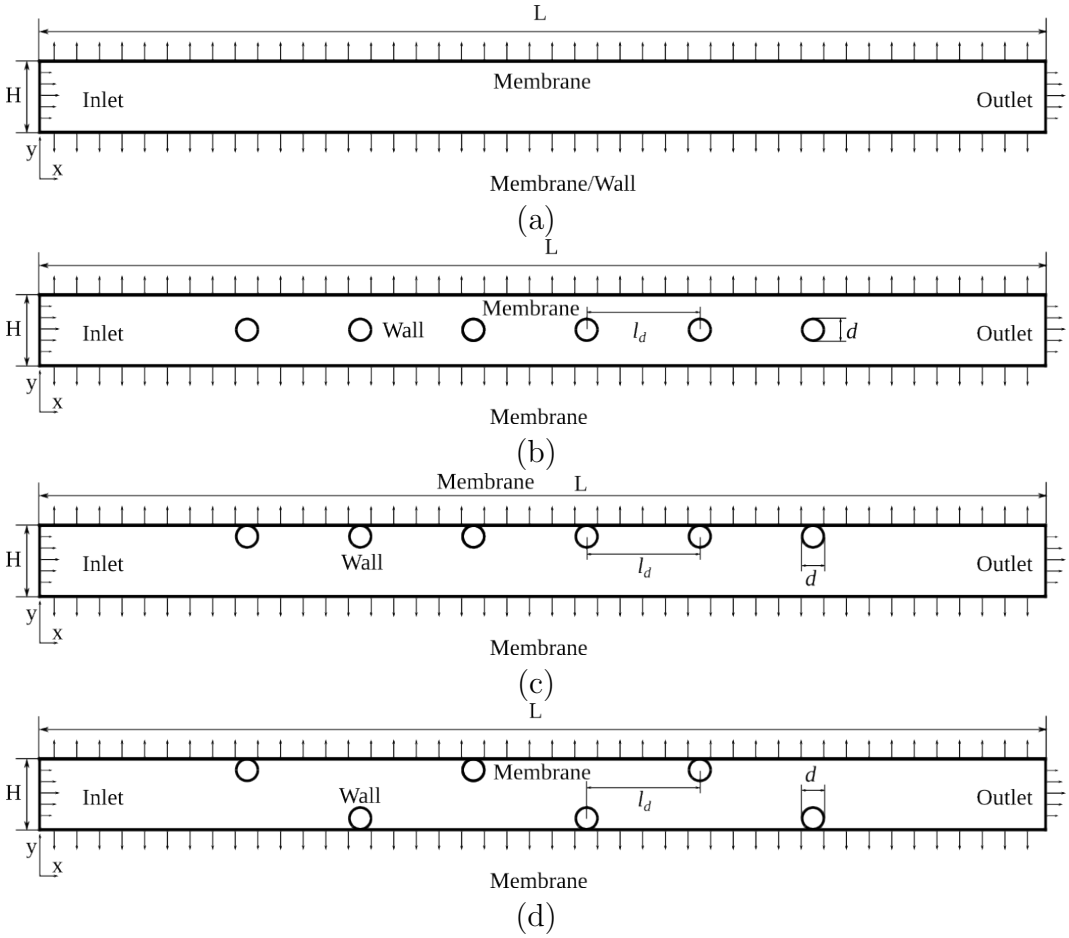


Figure 6.2: The flow channels considered in this study for the spacer-filled feed channel in a SWM module: (a) open channel, (b) submerged, (c) cavity and (d) zigzag spacers configuration

$L = 86.5mm$, height $H = 2mm$, spacers' diameter $d = 1mm$, interval distance $l_d = 8mm$.

6.2 Governing equations of RO process and boundary conditions

6.2.1 Governing equations

In order to solve membrane process flow problems, an additional equation for solute concentration is introduced to basic NS fluid equations. Governing equations of RO process for steady-state incompressible Newtonian flow are expressed as:

$$Contiuity \quad \nabla \cdot (\rho \mathbf{u}) = 0 \tag{6.1}$$

$$Momentum \quad \nabla \cdot (\rho \mathbf{u}\mathbf{u}) = -\nabla p + \nabla \cdot \left[\mu \left(\nabla \mathbf{u} + \nabla \mathbf{u}^T\right)\right] + \mathbf{f} \tag{6.2}$$

$$Concentration \quad Tranportation \quad \nabla \cdot (\rho \mathbf{u} C) = \nabla \cdot (\rho D \nabla C) \tag{6.3}$$

where D is the diffusion coefficient. During membrane precess, the changing solute concentration leads to changes in fluid density and viscosity. However, the change is small, so incompressible system assumption can be applied to this weak compressible fluid based on Boussinesq approximation [167]. Body force $\mathbf{f}$, like gravity, is normally ignored when there is no other external forces. Consider NaCl solution for example, which is the majority component of sea water. The correlation between concentration and density, viscosity and diffusion coefficient can be obtained via empirical physical properties as[168, 169]

$$\rho = 997.1 + 694C \tag{6.4}$$

$$\mu = 0.89 \times 10^{-3}(1 + 1.63C) \tag{6.5}$$

$$D = \max(1.61 \times 10^{-9}(1 - 14C), 1.45 \times 10^{-9}) \tag{6.6}$$

Consequently, six equations for six unknown variables guarantee a closed solvable system.

In this study, the thickness of the membrane is not resolved because of the insignificant dependence of the flow inside of the membrane on the external flows [169]. And hence, the membrane is considered as a two-dimensional plane. The roughness of the membrane surface is not considered and the surfaces are assumed to be smooth. Moreover, the water flux across the membrane depends on the concentration difference, the applied hydraulic pressure on the feed and the membrane properties.

Therefore, the water flux can be expressed as:

$$J_w = K(\Delta p - \Delta\pi) \tag{6.7}$$

where J_w is water flux, and this volumetric flux owns velocity unit and its value is the normal membrane boundary flow speed. K is the water permeability coefficient of the membrane; For RO, $\Delta p > \Delta\pi$. The osmotic pressure π is proportional to concentration C, for NaCl solution at 25 °C [169]:

$$\pi = 805.1 \times 10^5 C \tag{6.8}$$

6.2.2 Boundary conditions

Boundary conditions are listed in Tab. 6.1. Profiles of velocity and concentration are specified at the inlet and the zero gradient boundary conditions are applied for the velocity and concentration at the outlet. Spacers' surface is non-slip wall boundary. At the membrane surfaces, the tangential velocity is set to zero and the normal velocity is set to $u_{n,f} = J_w$. The boundary condition of concentration at the membrane surfaces can be derived based on the mass conservation of the solute. Using the rejection coefficient R which represents ratio of the rejected solute at the feed side, the concentration boundary condition can be obtained.

Table 6.1: Boundary conditions for RO model

Boundary	Boundary Condition
Inlet	$\mathbf{u} = \mathbf{u}_{in}$; $C = C_{in}$
Outlet	$\frac{\partial \mathbf{u}}{\partial n} = 0; \frac{\partial C}{\partial n} = 0$
Wall (non-membrane, non-slip)	$\mathbf{u} = \mathbf{0}$; $\frac{\partial C}{\partial n} = 0$
Membrane (feed channel)	$u_t = 0$; $u_{n,f} = K(\Delta p - C_{os}\Delta C_m)$
	$\rho u_{n,f} C R - \rho_f D \left(\frac{\partial C}{\partial n}\right)_f = 0$

6.3 Shape optimisation of the spacer

As mentioned earlier, spacers play two different roles, the mechanical support and the turbulence promoter. It obstructs the flow and significantly increases the shear

stress of the flow next to the membrane surfaces, which reduce the concentration polarisation and decrease the possibility of membrane fouling. However, there is a trade-off between the increase of the permeation and the increase of the pressure drop of the flow. Due to the obstruction of the spacers, high pressure drops of the flow are caused and lead to the higher energy consumption to desalinate water in a RO membrane process. A large amount of efforts have been made to optimise the spacers' shape for improving these two objectives. Spacer shape has different influences on these cost functions and the previous results indicate that spacer shape is more sensitive to affect the pressure drops through the membrane channel rather than the permeation rate [80]. But there is no clues to find the gradients of the spacer shape to optimise ether one of the two objectives. The gradient can provide the most sensitive parts of the spacer shape to affect the selected objective function, which helps understand the mechanism of spacers in the flow channels. Therefore, in this study, taking cylinder filament as a case study, an investigation to demonstrate the gradient-based method in the analysis and optimisation of the spacer shape aims to be carried out. Sensitivity analysis of all three spacer configurations is developed to identify the mechanisms of different placements affecting the pressure drop and permeation.

Before developing sensitivity analysis, the two objective functions are defined. The permeate flux is defined as the average membrane normal out flow velocity:

$$u_n = \frac{\int_{membrane} \mathbf{u} \cdot \mathbf{n} \mathrm{d}S}{\int_{membrane} \mathrm{d}S}. \tag{6.9}$$

The definition of mass averaged pressure loss is the same as Eq. (5.17) applied in Chapter 5. To develop a gradient-based optimisation, furthermore, pressure drop is selected as the objective function, because spacer geometry is more sensitively affecting the pressure drop through the flow channel.

6.4 Verification of the primal solver and the adjoint solver

6.4.1 Flow solver validation for channel flow with permeable wall boundary

Classic analytic model can be used for GPDE flow solver prediction test. For viscous flow in a 2D channel, pressure drop in a channel with impermeable walls can

be evaluated based on Darcy-Weisbach's law for Poiseuille flow[170]

$$\Delta p = \left(\frac{1}{2}\rho u_{avg}^2\right)\left(\frac{24}{Re}\right)\left(\frac{x}{h}\right) \tag{6.10}$$

where Reynolds' number $Re = \frac{\rho u_{avg}(4h)}{\mu}; 2h = H$; x is distance to inlet and H is the channel height. In a channel with two permeable walls (up and bottom walls), pressure drop based on Berman's solution [171, 172] is determined analytically:

$$\Delta p = \left(\frac{1}{2}\rho u_{avg}^2\right)\left(\frac{24}{Re} - \frac{648}{35}\frac{Re_w}{Re}\right)\left(1 - \frac{2Re_w}{Re}\frac{x}{h}\right)\left(\frac{x}{h}\right) \tag{6.11}$$

where Re_w is Reynolds number at permeable boundary, and $Re_w = \frac{\rho u_w h}{\mu}$. The geometric domain is open channel with height $H = 0.001m$ and length $L = 2m$. Inlet velocity is $0.1ms^{-1}$ and Reynolds number for channel flow is $Re = 200$ with fluid density $\rho = 1000kgm^{-3}$ and fluid dynamic viscosity $\mu = 10^{-3}Pas$. For permeable wall, constant permeation velocity of $8 \times 10^{-6}ms^{-1}$ is applied. GPDE simulates channel flow with a fully developed inlet velocity:

$$u = 6u_{avg}\frac{y}{h}\left(1 - \frac{y}{h}\right) \tag{6.12}$$

and computation domain is 1mm channel height × 2m channel length. Comparison among Poisruille's solution, Berman's solution and GPDE solution of pressure drop along the axial channel direction is shown in Fig.6.3.

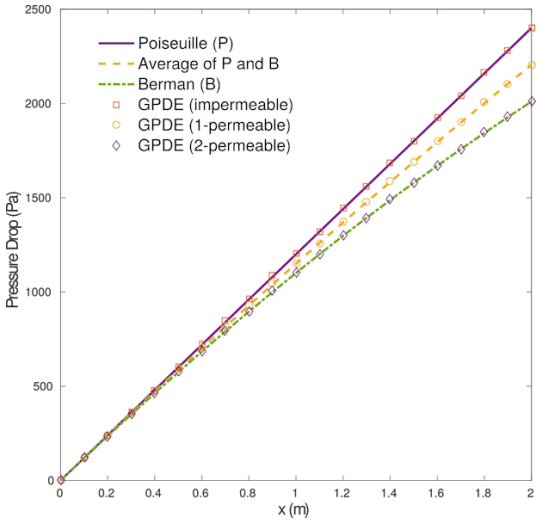


Figure 6.3: Pressure drop comparison among GPDE and analytical solutions

6.4.2 RO process computation validation

For reverse osmosis membrane process, cross velocity is influenced largely by solute fraction for a given constant operational pressure.

$$u_{cross} = u_{n,f} = J_w \tag{6.13}$$

Normally in the channel for RO model, feed flow has much higher solute fraction ($C_{feed} > C_{permeate}$), so the cross velocity is mainly determined by the feed side solution fraction, which also allow two-channel computation to be simplified as single channel model simulation. Solution from GPDE is compared with Fletcher's numeri-

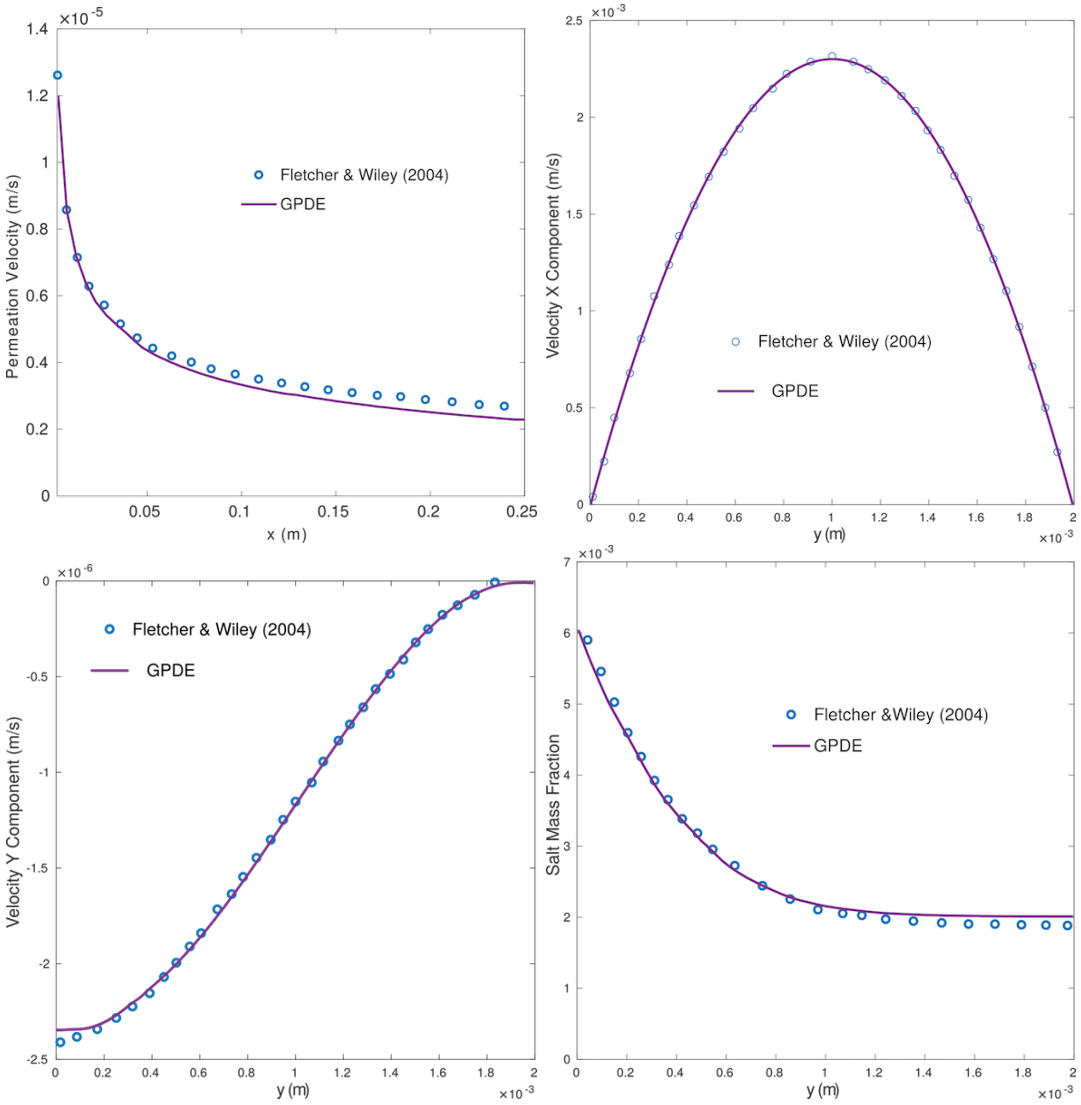


Figure 6.4: Comparison with Fletcher's results: cross velocity profile along membrane (up left), x velocity component profile along y direction at $x = 240mm$(up right), y velocity component profile along y direction at $x = 240mm$(down left) and salt mass fraction profile (down right)

cal results [173] in Fig. 6.4. From the matching results, the flow solver for RO model

is verified. The simulated cross membrane permeate flux using GPDE is validated with experimental data [174] as shown in Fig. 6.5.

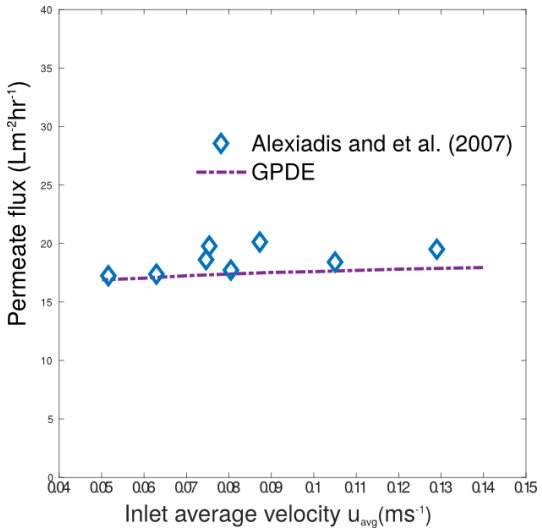


Figure 6.5: Comparison of cross membrane permeate fluxes between GPDE's solution and experimental data

6.4.3 Verification of adjoint sensitivity for RO cases

Additional convention-diffusion equation and new membrane boundary condition are introduced to the coupled flow system, so primal solve is re-differentiated. Sensitivity validation is conducted via comparison among FD, Tangent linear solution and adjoint solution. Flow in open channel in Fig. 6.2 (a) with one wall, one membrane, one inlet and one outlet boundaries is computed as the sensitivity validation case. Inlet velocity distribution satisfies fully-developed profile:

$$u = 6u_{avg}\frac{y}{h}\left(1-\frac{y}{h}\right), \quad v = 0. \tag{6.14}$$

where the inlet average velocity value u_{avg} is 0.1 ms^{-1}. Channel height H is $2mm$ and length L is $86.5mm$. NaCl solution as test fluid has dynamic viscosity $\mu = 0.89 \times 10^{-3} Pas$ and density $\rho = 997.1 kgm^{-3}$. Operational pressure on membrane is $898.7kPa$. Rejection coefficient is chosen as 99.5%. Water permeate coefficient of membrane K is $6.93 \times 10^{-12} ms^{-1} Pa^{-1}$ [174]. Initial mass fraction of salt is 0.002. Control variables are coordinates in computational domain. Average permeate flux is the objective function. Sensitivity values via three methods are shown in Tab. 6.2. Choosing tangent linear result as reference, FD with step-size $\Delta x = 3.5 \times 10^{-7}$ has the smallest absolute error as 4.455×10^{-10} and discrete adjoint sensitivity's absolute error achieves 10^{-17} which is considered as accurate as machine precision.

Flow fields and impermeable wall surface normal sensitivity are shown in Fig. 6.6.

Table 6.2: Finite difference solution compared with tangent linear and discrete adjoint

Method	Sensitivity	$Error_{abs}$
FD	-8.6054939472090856E-007	4.455073084104544e-10
Tangent	-8.6099490202931902E-007	0.0
Adjoint	-8.6099490195082638E-007	7.849263739792469e-17

Because membrane is not suitable for deforming, wall boundary is chose as movable surface where the surface sensitivity is our concern.

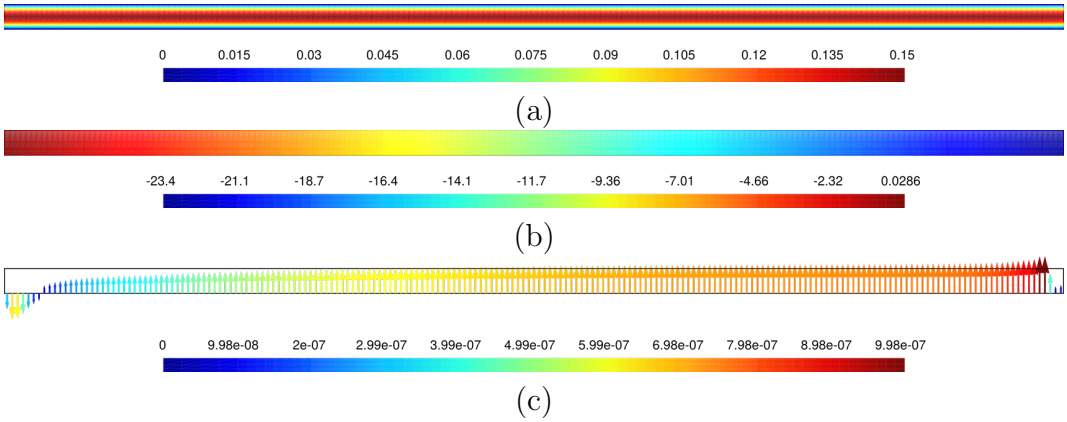


Figure 6.6: Open channel solution: (a)velocity magnitude distribution, (b)pressure field and (c) impermeable wall surface normal sensitivity of permeate flux

6.5 Flow patterns of the spacer-filled feed channels in the RO membrane process

At first, in order to carry out the sensitivity analysis of the spacer shape with respect to the two objectives, the flows in all the three spacer configurations are simulated. Boundary refined non-uniform mesh is generated to achieve accurate evaluation solutions, and the height of the membrane side first layer cells' is about $5\mu m$. Mesh-independent computation is guaranteed with more than 130k cells used for all cases, which meets the mesh-independent requirement according to previous work [175]. Tab. 6.3 shows objective functions' values among different arrangements. Scaling Reynolds number is based on the inlet velocity and the height of channel:

$$Re = \frac{\rho u_{avg}(2H)}{\mu} \tag{6.15}$$

where $\rho = 997.1kg/m^3$, $\mu = 0.89 \times 10^{-3}Pas$, $H = 0.002m$, $u_{avg} = 0.01, 0.05, 0.1$ and $0.2ms^{-1}$ respectively for cases comparison. The results of the velocity streamline

Table 6.3: Objective function comparison among Reynolds number

Configuration	Re=44.8		Re=224		Re=448		Re=896	
	$u_n(ms^{-1})$	$\Delta p(Pa)$	$u_n(ms^{-1})$	$\Delta p(Pa)$	$u_n(ms^{-1})$	$\Delta p(Pa)$	$u_n(ms^{-1})$	$\Delta p(Pa)$
Open channel	4.343E-6	2.269	4.683E-6	11.52	4.778E-6	23.08	4.851E-6	46.20
Cavity	4.619E-6	4.129	4.876E-6	29.85	4.940E-6	74.26	4.994E-6	187.5
Submerged	4.524E-6	7.299	4.832E-6	49.74	4.918E-6	130.2	4.991E-6	358.5
Zigzag	4.643E-6	4.164	4.885E-6	29.67	4.950E-6	74.89	5.000E-6	207.9

in the cavity, submerged and zigzag configurations are shown in Fig. 6.7, Fig. 6.8 and Fig. 6.9 respectively where the flow field of the third and the fourth filaments are plotted.

The feed flow is significantly changed by placing spacers in the membrane channel. The acceleration and deceleration of the flow caused by the spacers' volume and position create pressure variations. The adverse pressure gradient which is caused by the deceleration near the rear of spacer, results in flow separation, recirculation or/and vortex shedding. The feed is forced to flow through the obstructed area and then expands to refill the flow area behind the spacer. As a consequence, rapid velocity variations are caused, which lead to viscous dissipation of the flow momentum. Generally, two mechanisms of transfer enhancement are applied by the spacers: 1) flow separation caused by the spacers on the membrane surfaces perpendicular to the bulk flow direction, and the resulted flow re-attachment and re-development of boundary layers downstream of the spacer along the membrane surfaces; 2) high shear stress of the flow to the membrane surfaces due to the increased flow velocity.

Although both mechanisms are involved in all the three studied configurations, the major contribution to the enhancements varies in different spacer configurations. In the cavity configuration, as shown in Fig. 6.7, stagnant zones are found in front and behind each filament. When the flow velocity is slow, the slowly rotating vertices are generated after the spacer next to the membrane surface. Due to these vertices next to the spacers downstream lead to disruption of the built-up CP layers and re-attachment of the flow and concentration boundary layers, resulting in the enhancement of permeation. In the flow with low Reynolds number, the vertices themselves have little contribution to the mass transfer enhancement, which are usually called "dead zones" [176]. In these zones, the feed undergoes slowly circulatory motion with little mixing from the bulk feed flow. Salt movement from bulk to dead zone is mainly by diffusion in which convective transfer arising out of feed flow has little contribution. Therefore, salt concentration slowly gets depleted off and the CP boundary layer increases. When the Reynolds number of the flow increases, a stretching of a time-averaged recirculation zones occurs. In the high Reynolds number

flows, the unsteadiness of the shedding vertices occur and enlarge the recirculating zones. The mixing of the flow is achieved by the successive shed vortices behind the upstream spacer continuously mixing and merging before the downstream filament. Consequently, the overall mass transfer is enhanced. In addition to the formation of the vertices at the membrane surface with the filaments, the accelerated flow at the opposite membrane surface also improve the mass transfer across the membrane.

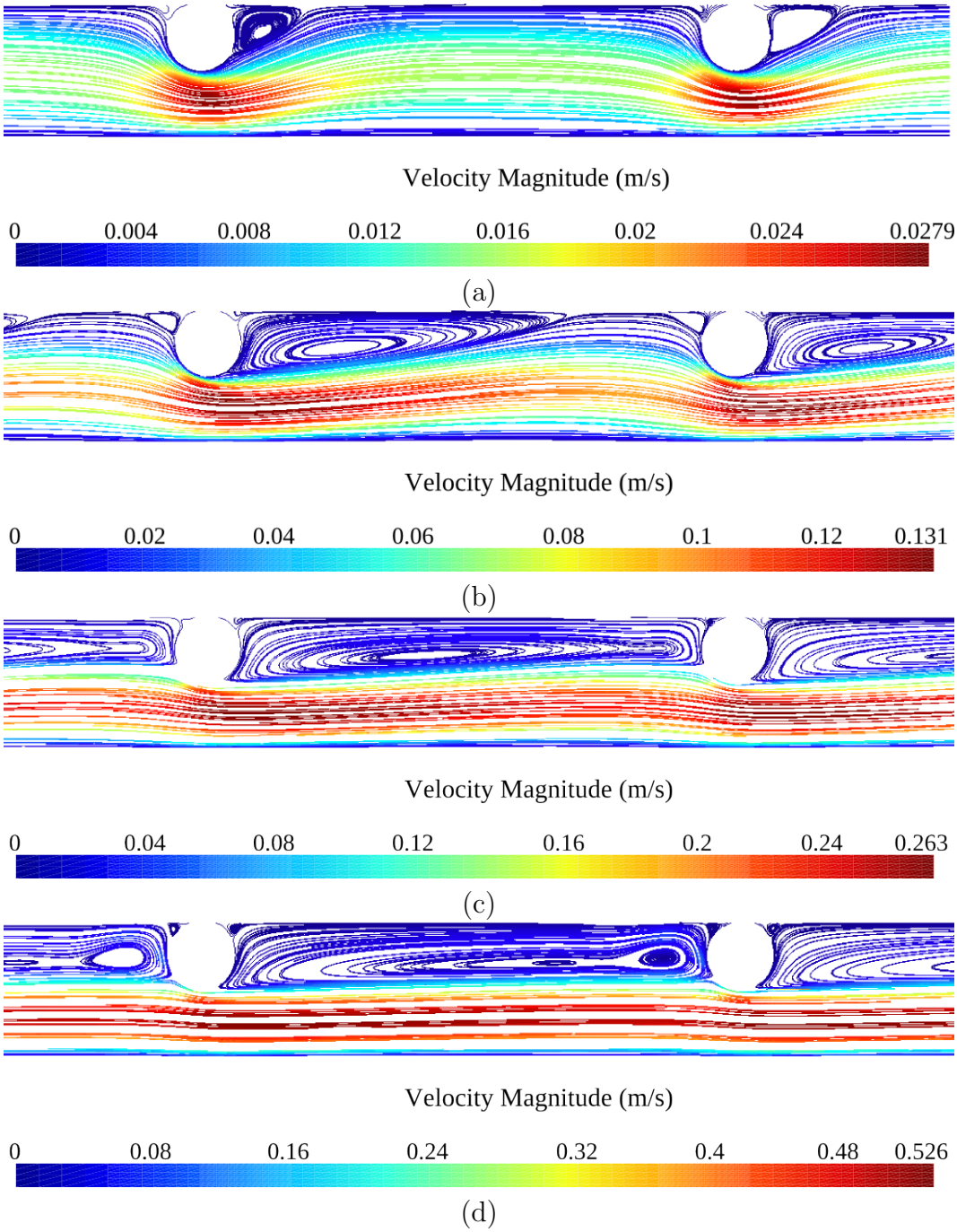


Figure 6.7: Streamlines of cavity configuration at Re=44.8 (a), 224 (b), 448 (c) and 896 (d)

Furthermore, as shown in Fig. 6.8 in the submerged configuration, the cross-section area starts to decrease with the leading edge of the spacers placed in the middle of the channel. High velocity of flow is generated due to the further decreased cross section flow area in the membrane channel, which will increase scouring force on the membrane wall. These phenomena improve the generation of drag effects and deter the formation of a thick concentration boundary layer near the membrane wall. Thus, high value of local velocity is the main reason to reduce the CP and improve the mass transfer across the membrane surfaces in this spacer configuration. As shown in Fig 6.8, as the Reynolds number of the feed flow increases, the time averaged recirculation area increases. The submerged configuration leads to higher velocity of the flows at the membrane surfaces in a larger region with the reduced cross-section area between every two consecutive filaments.

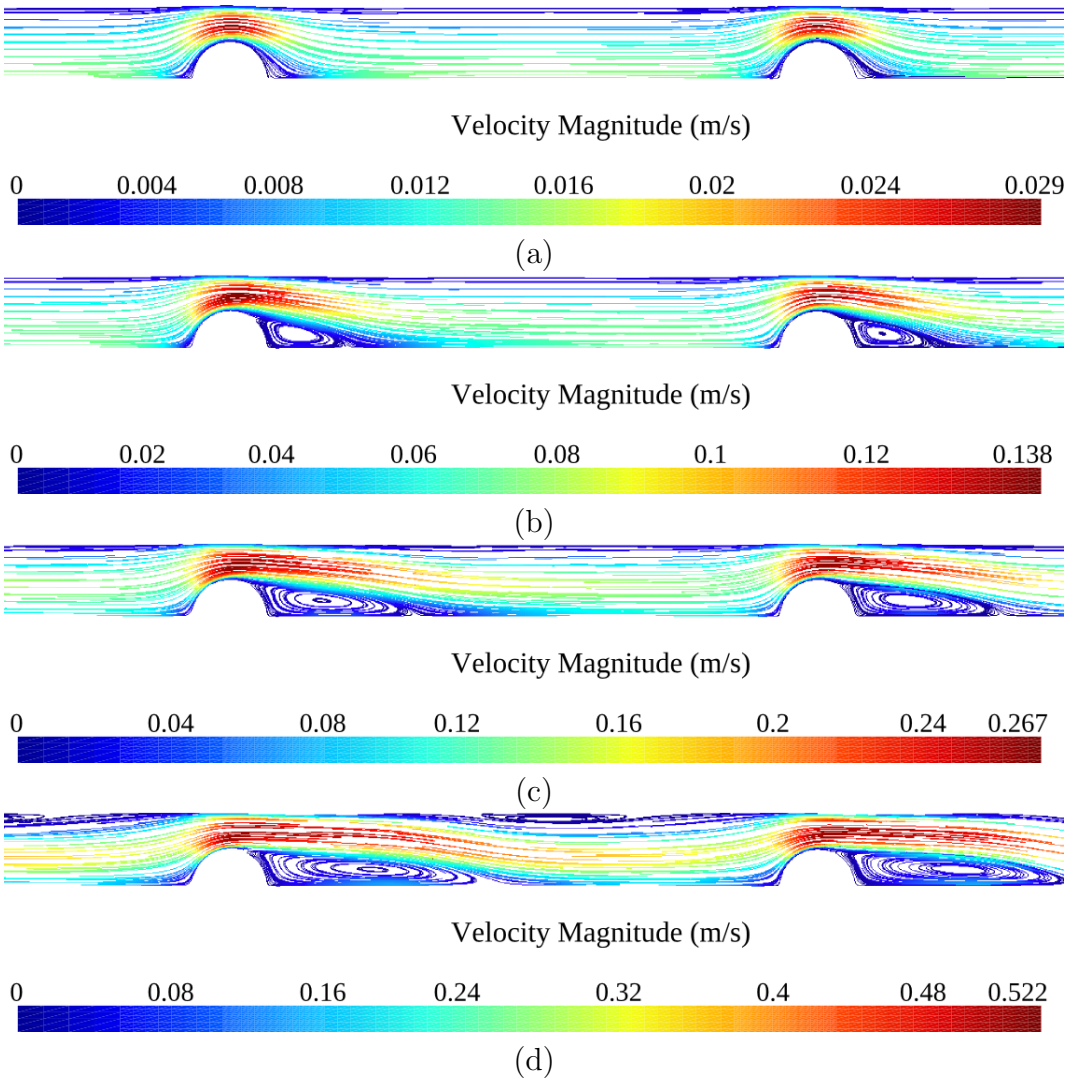


Figure 6.8: Streamlines of submerged configuration at Re=44.8 (a), 224 (b), 448 (c) and 896 (d)

According to the flow streamlines in the zigzag configuration, as shown in Fig. 6.9, dead zones also exist in front and behind the filaments. But different from the flow in the cavity configuration, the bulk stream is forced to flow in a more tortuous path. Also, in every membrane surface, mass transfer enhancement methods of both the flow separation and the increased flow shear rate take place. With the increase of the Reynolds number, the time-averaged recirculation area becomes larger and larger, leading to the increased regions with the reduced cross-section flow area. As a result, higher shear stresses are applied to larger area at the membrane surfaces.

Besides, the pressure drop is also a major concern in the membrane channel flows. In an open channel flow, the pressure drop is caused by the viscous effect of the feed. And in a spacer-filled channel, much higher pressure drop is due to the momentum losses when the feed flow is obstructed by the spacers and the abrupt changes in the flow and the formation of drag. According to the results, significant pressure drops are caused in all the spacer-filled flow channels compared to that in the open flow channel. Furthermore, when the pressure drop is compared among the different spacer configurations, it clearly indicates that the highest pressure drop occurs in the submerged configuration and pressure drops are relatively close in the cavity and zigzag configurations, according to the results with different Reynolds numbers. Because the spacers are placed in the centre of the flow channel, the highest velocity is generated which leads to the abrupt momentum changes and the highest pressure drop. In contrast, because of the identical size and shape and similar placement of the spacers (next to the membrane surface) in the cavity and zigzag, the pressure drops of both configuration are close. The slight differences of the values are due to the different flow paths of the feed flow and the coupling effect with the increase of the velocity. In the cavity configuration, the feed flow takes the less tortuous path. While more flow changes in flow direction occur in the more tortuous path in the zigzag configuration.

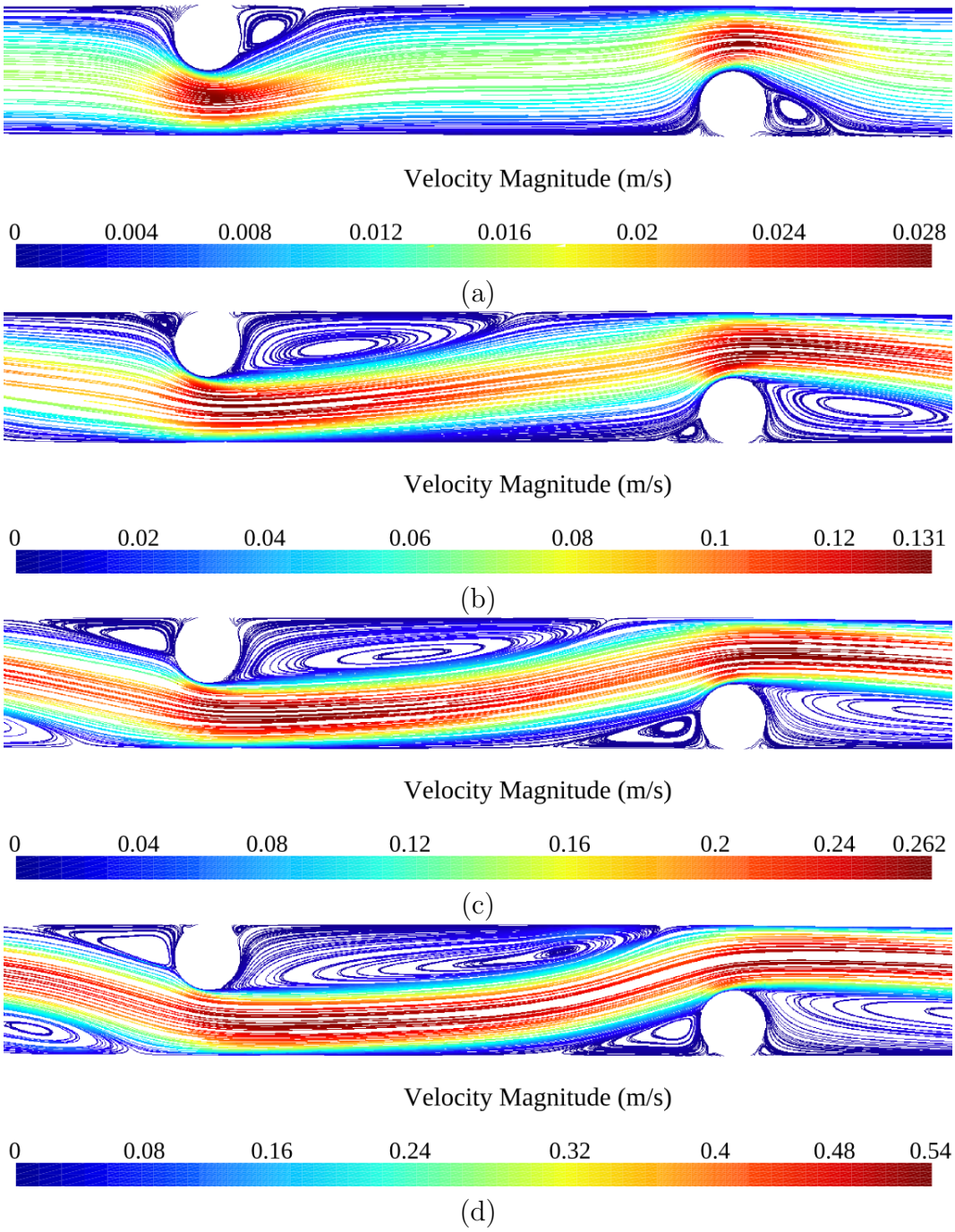


Figure 6.9: Streamlines of zigzag configuration at Re=44.8 (a), 224 (b), 448 (c) and 896 (d)

6.6 Sensitivity analysis of the spacer shape in the feed channel using discrete adjoint

Based on the analysis of the flow patterns in different spacer configurations, it clearly indicates the mechanisms of the placement of the spacers to improve the permeations and the trade-off relationship between the permeation and the pressure drop. But it is difficult to further analysis the contributions of the different influencing factors of spacers in any configuration. For example, in a zigzag or cavity configuration, it is hard to identify whether the influence of the stagnant region in front of the filament is larger or the one behind is more important for the improvement. From a particular flow field of any spacer configuration, it is difficult to identify which mass transfer enhancement has larger contribution. Therefore, in this section, a sensitive analysis is carried out to identify the most sensitive factors of the cylindrical filament in different spacer configuration with different Reynolds numbers.

Gradients of the spacer shape of the cavity configuration with respect to the permeation and the pressure drop are shown in Fig. 6.10 and Fig. 6.11 respectively, in which the third and the fourth filament in the four flows with different Reynolds number are considered. The gradients field is normalised for every simulation case to indicate the sensitivity distribution.

According to the gradient profiles of the spacer shape with respect to the permeation, as shown in Fig. 6.10, the most sensitive parts to determine the performance of the mass transfer is the stagnant zones both in front and behind the filaments. In all the four studied Reynolds numbers, the highest gradient fields locate at the parts of the filament close to the membrane surface. The results demonstrate the more contribution of the flow separation which cuts the built-up CP layer and leads to the re-attachment of the flow and boundary layer to the mass transfer enhancement in the cavity configuration. Moreover, with different Reynolds number, the gradient profile changes slightly. In the lower Reynolds number, as shown in Fig. 6.10(a) and Fig. 6.10(b), a more sensitive of the stagnant zone in front of the filament contributes to the permeation. In contrast, as shown in Fig. 6.10(c) and Fig. 6.10(d), the more sensitive parts move to the stagnant region behind the filament.

Compared with the permeation surface sensitivity, the gradient profiles of the spacer shape in the cavity configuration indicate the different distribution with respect to the pressure drop. As shown in Fig. 6.11, the most sensitive parts of the spacer shape are the lower surface of the filament which is far away from the membrane. The gradients are toward the direction to decrease the pressure losses due to the eddy formation in the separated flow. In addition, with the increase of the Reynolds number, the sensitive parts of the shape move slightly close to the apex of

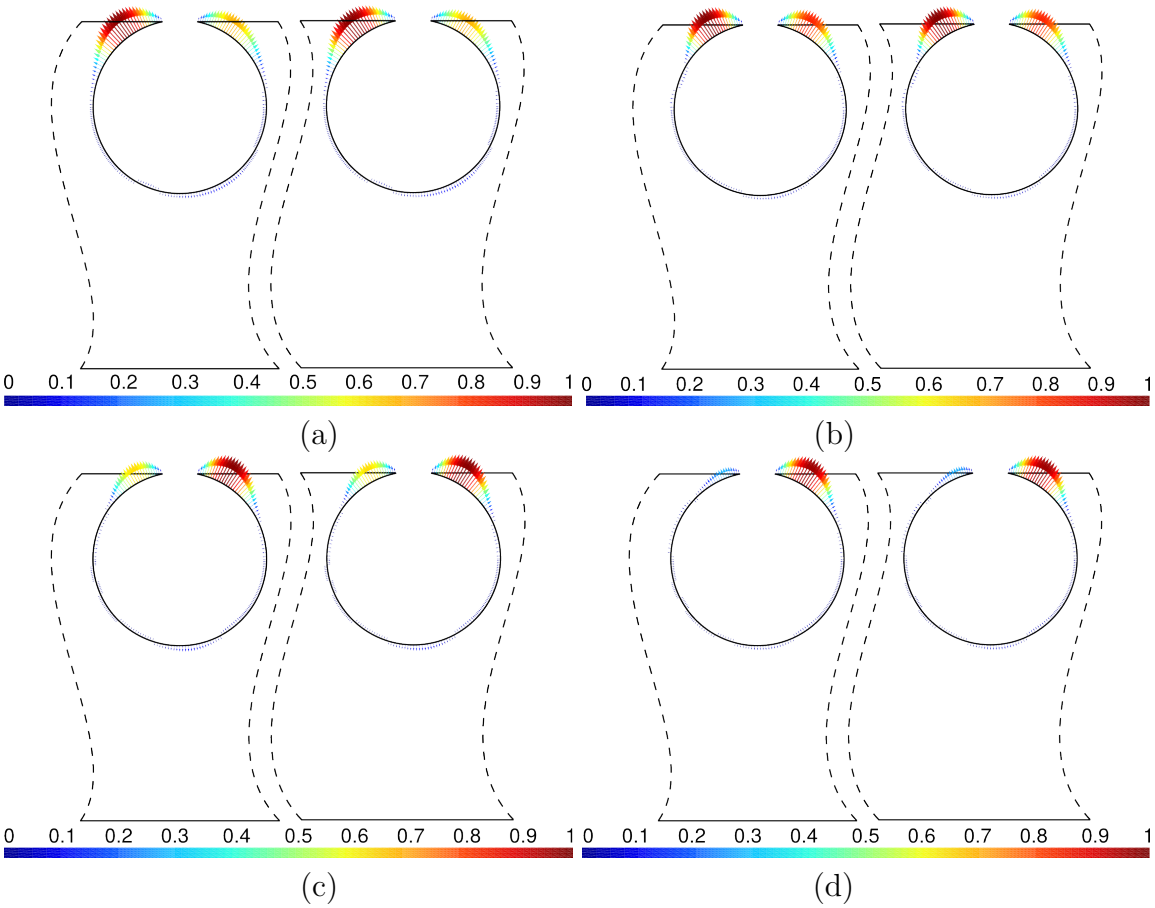


Figure 6.10: Permeate flux sensitivity of cavity spacers at Re=44.8 (a), 224 (b), 448 (c) and 896 (d)

the low surface of the filaments. The reasons to these facts are that the gradients are generated according to the flow pattern of the cavity configuration to generate a more streamlining spacer. Based on the streamlines plotted in Fig. 6.7, with the increase of the Reynolds number more flat streamlines are produced in the flow channel, which lead to the concentrating gradient toward the apex of the low surface of the filament.

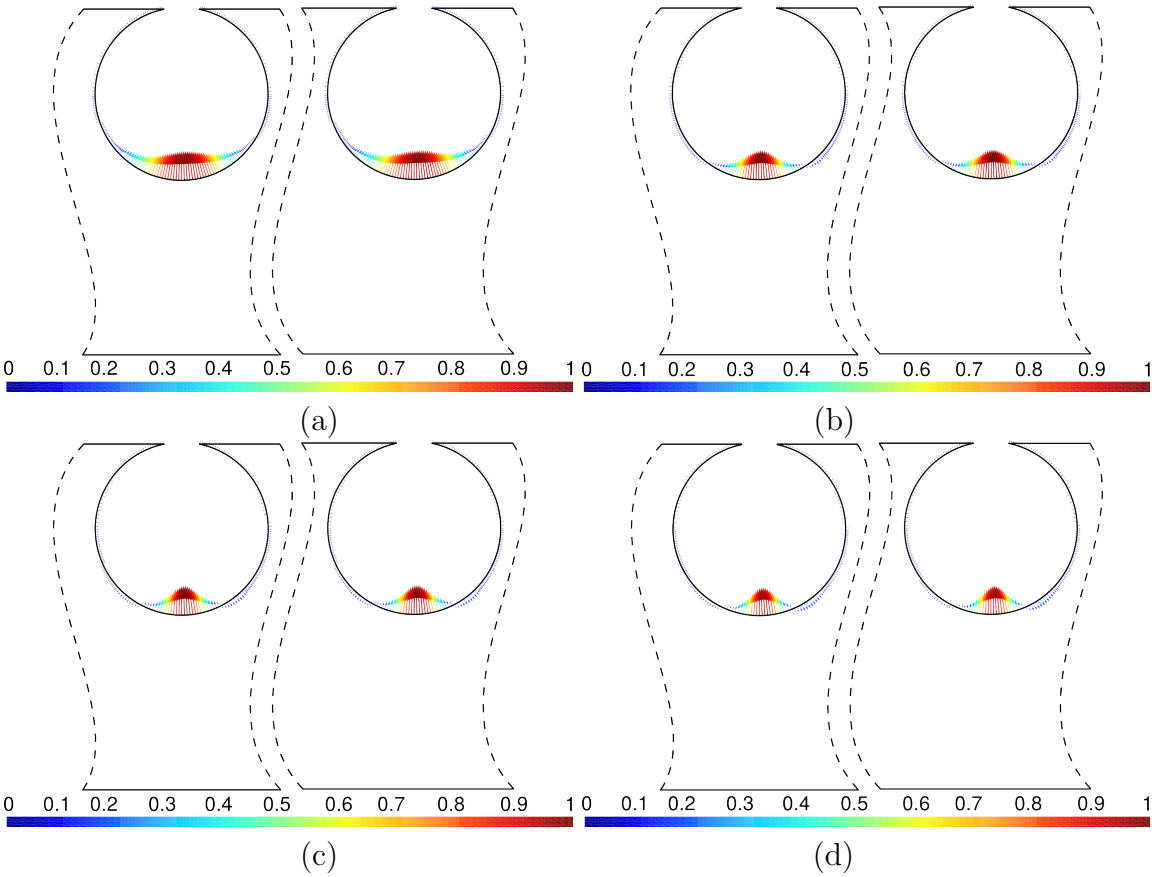


Figure 6.11: Pressure drop sensitivity of cavity spacers at Re=44.8 (a), 224 (b), 448 (c) and 896 (d)

The gradient field of the spacer shape in the submerged configuration with respect to the permeation and the pressure drop are shown in Fig. 6.12 and Fig. 6.13 respectively in which four flows with different Reynolds numbers are considered.

For improving the permeation, compared to the cavity configuration, as the filaments are placed away from the membrane surface, the sensitive parts in the submerged are the upper and lower surfaces of the filament rather than the stagnant zones in front and behind the filament. Similar trends of the gradient field are found the submerged configuration to improve the pressure drops. Furthermore, with the increase of the Reynolds number the most sensitive parts are shifted to the apex of the surfaces. In the low Reynolds number, according to the flow pattern as shown in Fig. 6.8, there is no obvious flow separation and the flow is closely attach to the

cylinder filament surface. The most sensitive parts to improve the permeation are the regions in front and behind the apex of the surfaces, which prolong the high velocity region in both directions in the flow channel. With the increase of the Reynolds number, flow separation behind the filament occurs and becomes more and more strengthened. Thus, the gradient profile of the spacer shape is toward the direction to enhance the flow separation for improving the permeation. As shown in Fig. 6.12, the region behind the apex becomes more sensitive in the flow with Reynolds number 224 and gradually moves to the apex when the Reynolds number increases to 448 and 996.

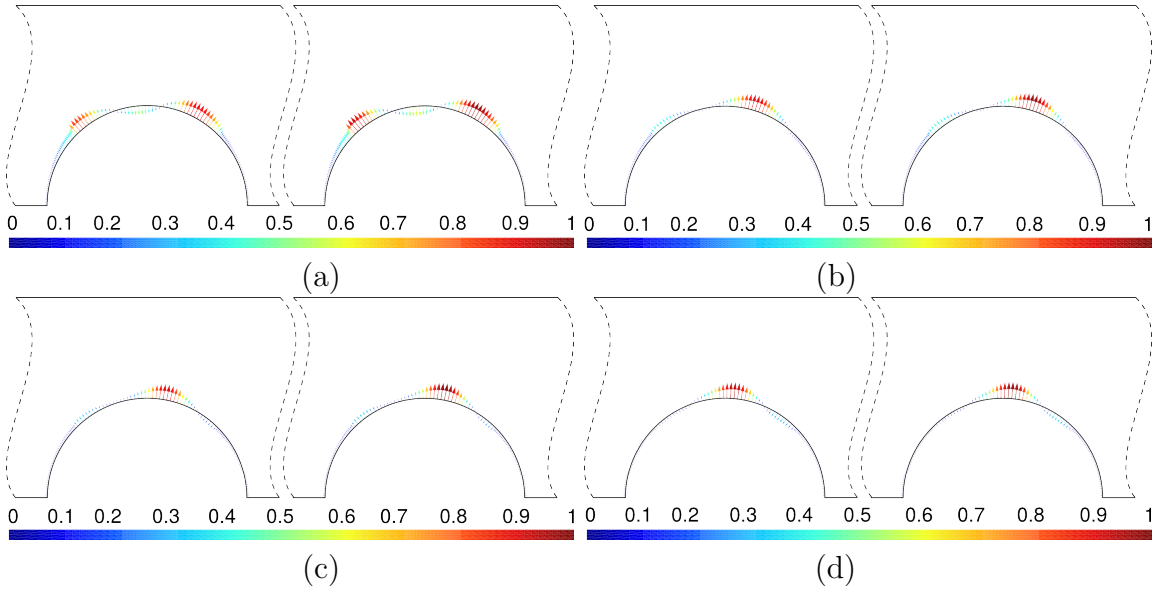


Figure 6.12: Permeate flux sensitivity of submerged spacers at Re=44.8 (a), 224 (b), 448 (c) and 896 (d)

On the other hand, in order to decrease the pressure drop, the gradient fields is more uniformly distributed in the flow with Reynolds number 44.8 because of the closely attached flow in the upper and lower surfaces of the filament. While with the increase in Reynolds number, the most sensitive parts move to the apex of the upper and lower surfaces to reduce the pressure drags due to the flow separation behind the filaments. This gradient distribution shift can be found in Fig. 6.13.

The gradient field of the spacer shape in the zigzag configuration with respect to the permeation and the pressure loss are shown in Fig.6.14 and Fig. 6.15 respectively in which the flows with four Reynolds number are plotted.

For maximising the permeation in the zigzag configuration, the sensitive parts of the spacer shape are different from both the cavity and submerged ones. The large gradients are in both the stagnant zones and the upper and lower surfaces. Additionally, with the increase of the Reynolds number, on one hand, similar to the cavity, the sensitive parts of the stagnant parts move from the front one to the one

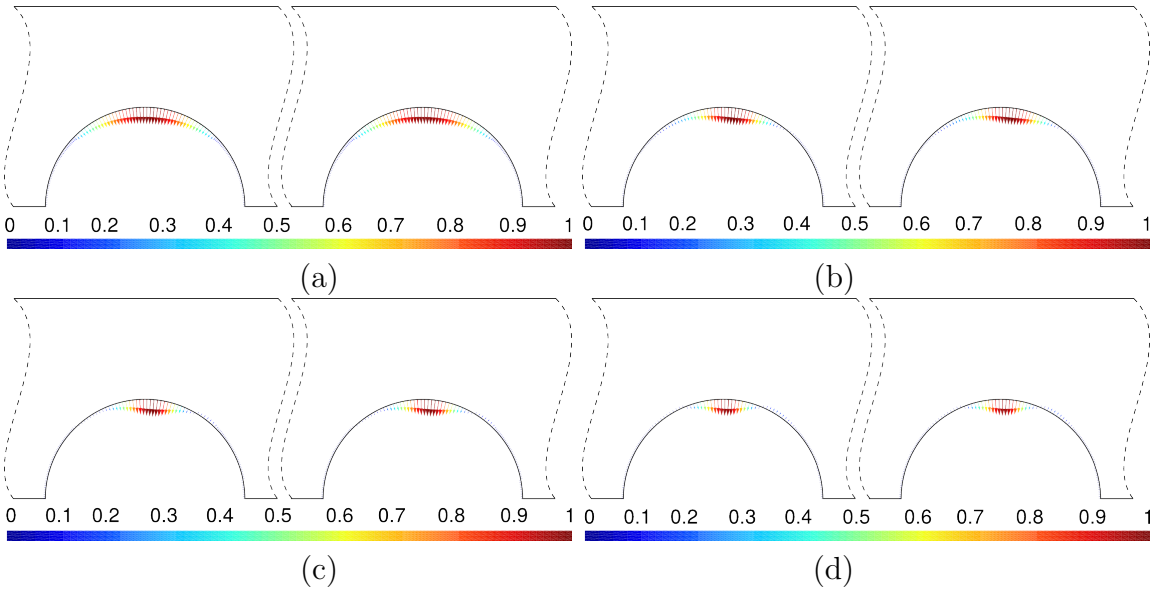


Figure 6.13: Pressure drop sensitivity of submerged spacers at Re=44.8 (a), 224 (b), 448 (c) and 896 (d)

behind the filament. On the other hand, the sensitive parts of the upper and lower surfaces are shifted to the surfaces' apexes in the filaments. Although the values of the gradients of the shape at the stagnant zones are larger than the ones of the shape at the upper and lower surfaces, it indicates the zigzag configuration has both the discussed mechanisms to improve the permeation, which not only disrupts the concentration boundary layer on the membrane surface the filaments located, but also increase the velocity of the flow at the opposite membrane surface.

In addition, the gradient field of the shape to decrease the pressure drops in the zigzag configuration is also similar to the previous studied two configurations. As shown in Fig. 6.15, the most sensitive parts of the shape are the upper and lower surfaces. Therefore, from the results of the three configurations, it indicates that the gradients of the spacer shape with respect to the pressure loss are less influenced by the spacer configuration.

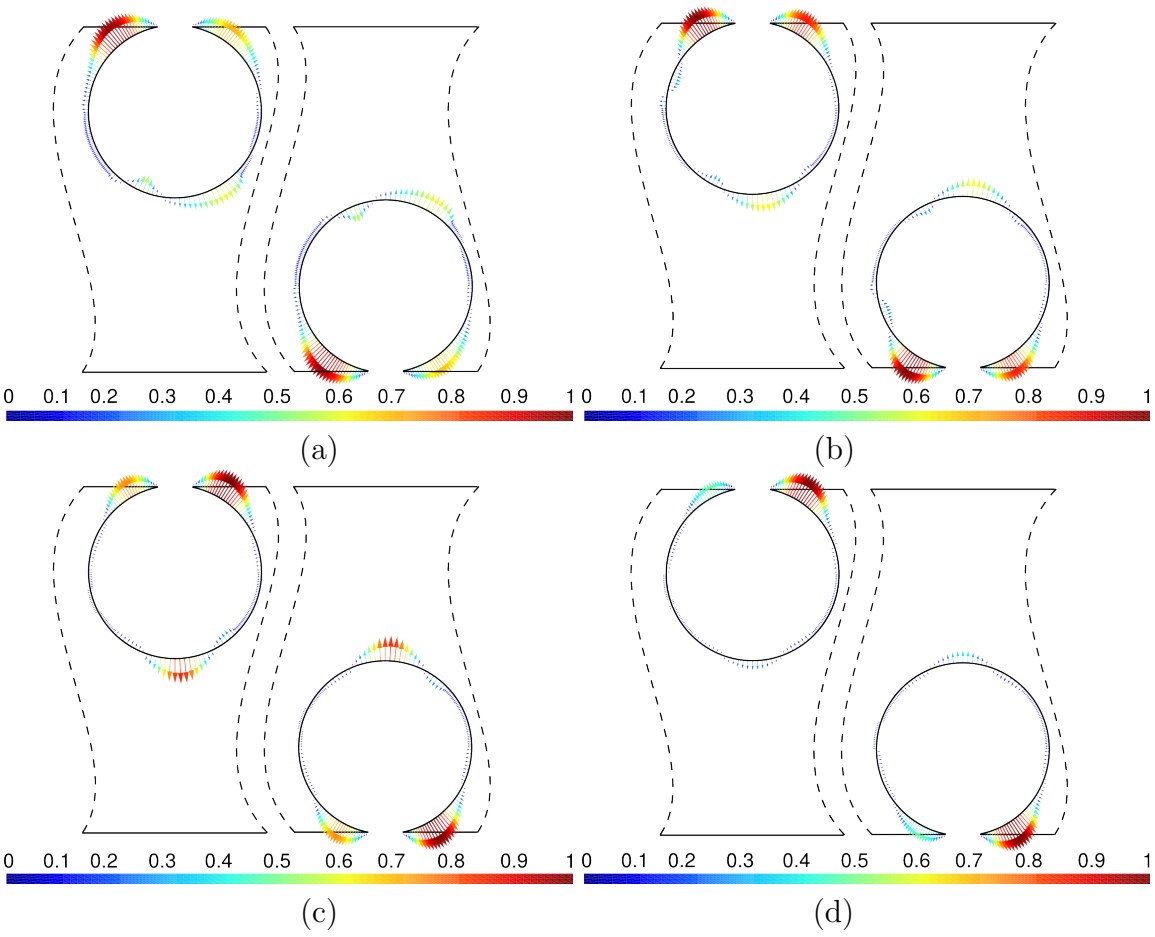


Figure 6.14: Permeate flux sensitivity of zigzag spacers at Re=44.8 (a), 224 (b), 448 (c) and 896 (d)

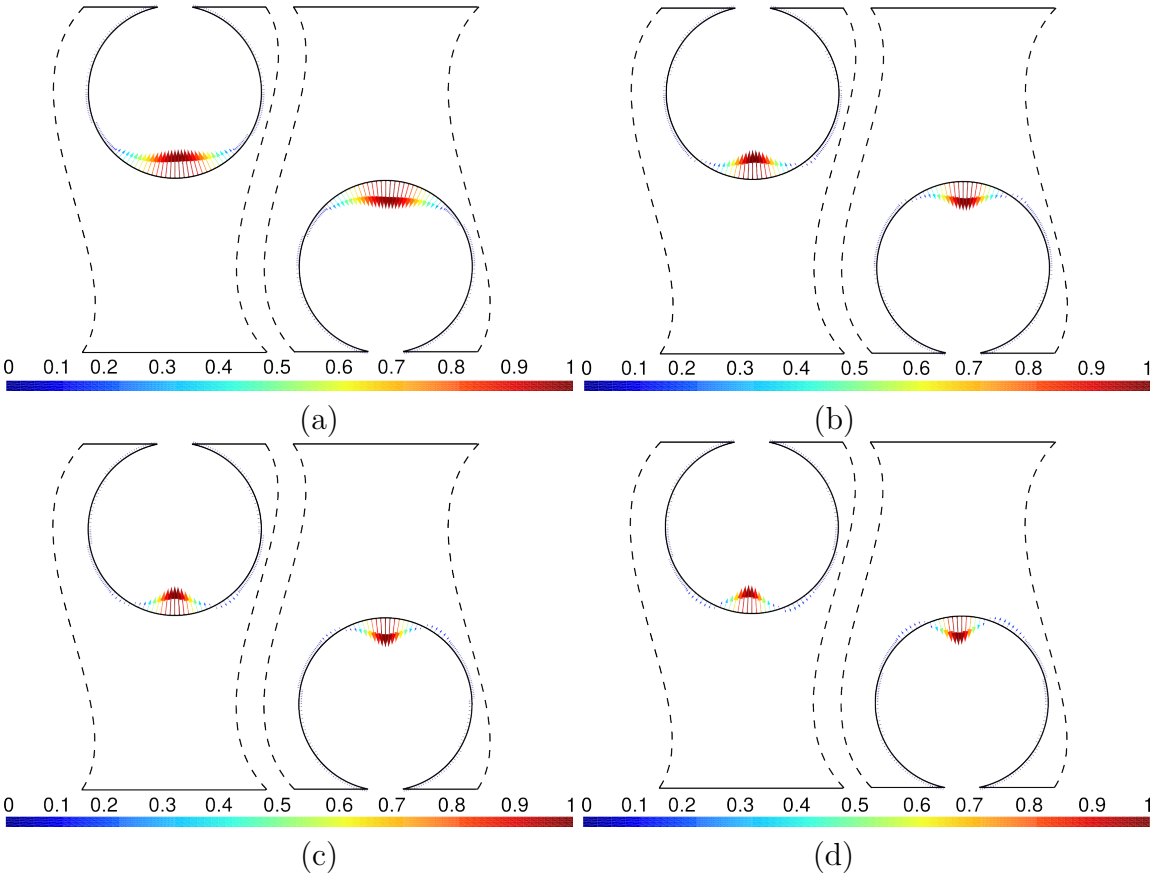


Figure 6.15: Pressure drop sensitivity of zigzag spacers at Re=44.8 (a), 224 (b), 448 (c) and 896 (d)

6.7 Spacers' shape optimisation in RO membrane channel

From the literature, it is indicated that spacer geometry more sensitively affects the friction-related hydraulic pressure losses compared to the mass-transfer related osmotic losses [80]. From the evaluation of objective function results in Tab. 6.3, it also can be found the larger influence of Reynolds numbers on the pressure drop than the permeate flux. In this context, an optimisation to minimise pressure drop is developed based on adjoint method. At the early stage, the shape of spacers with zigzag arrangement at Reynolds number 44.8, as a case study, is optimised.

With the sensitivity calculated using discrete adjoint method, back-tracking line search with Armijo-Goldstein condition is applied to drive this gradient-based optimisation loop. As the surface sensitivity distribution shown Fig. 6.15, the displacement based on the gradient is smooth. No additional smoothing is required for this case. Through about 90 iterations of optimisation, the obtained optimum shape is shown in Fig. 6.16, in which the dash line illustrates the optimised shape for reducing pressure drop. The optimisation convergence of objective function and the gradient is shown in Fig. 6.17. Based on this optimised spacers' shape, the pressure drop though the flow channel reduces by 24%, and the permeate flux reduces by 0.43%.

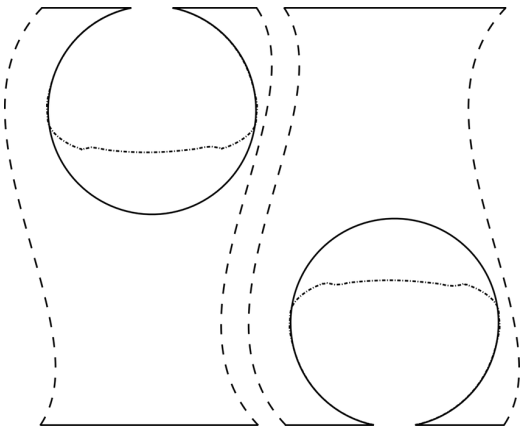


Figure 6.16: Pressure drop optimisation shape change

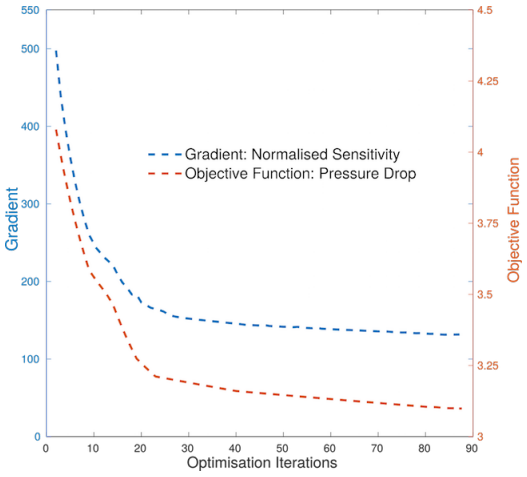


Figure 6.17: Optimisation convergence of the pressure drop

The streamlines comparison between these two shapes are illustrated in Fig. 6.18 in the same velocity magnitude scale. All computation is based on the same inlet mass flow assumption. The squeezed spacers enlarge the flow channel area, and then the velocity passing through the narrow segment of the flow channel reduce to conserve the mass flow. At this Reynolds number, pressure drop comes from skin friction mainly. And the reduced the velocity gradient leads to smaller shear stress, which results in the reduced friction force. As it is shown in Fig. 6.18, the down

stream vortices after the optimised cylinder is slightly smaller than the initial flow state. The flow mixing status doesnt change much, so the permeate flux reduces slightly.

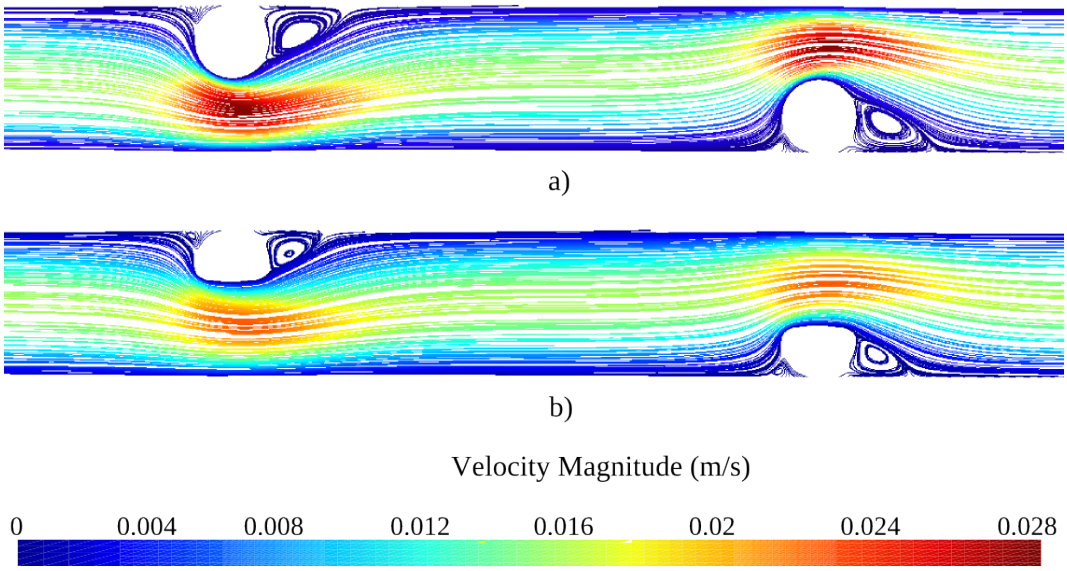


Figure 6.18: Streamlines comparison between initial and optimised flow fields

Chapter 7

Conclusion and future work

7.1 Conclusion

In this study, the development of incompressible flow and discrete adjoint solvers is based on in-house code GPDE, where SIMPLE algorithm is used and discrete adjoint solver is implemented via automatic differentiation tool Tapenade. The instability issues of standard SIMPLE algorithm occurs in certain practical cases. As a result, the discrete adjoint solver can diverge or fail to calculate the accurate sensitivity when the primal diverges or only converges to LCO. The efficient and accurate sensitivity solution depends on the improvement of the primal solver, which is essential for the gradient-based shape optimisation in practical industrial applications. Therefore, this thesis aims to achieve a more robust and stable performance discrete adjoint solver by improving stability of the flow solver. Theories of stability strategies are studied and the performance is demonstrated through case studies.

Via revisiting SIMPLE-like algorithms, pressure Schur complement approach is introduced and derived in detail to build the bridge between this mathematical method and classical "prediction-correction" point of view of SIMPLE-family algorithms. Four PSC schemes are introduced and their robustness are compared. Among them, SAI shows the most stable performance since the same convergence can be obtained with changing pressure under-relaxation factors in its work space; "neighbour-correction" PSC inspired SIMPLEC gives the most robust performance as the largest work space for pressure under-relaxation factor.

Skewness correction methods based on FVM are important for reducing the instability due to the low quality of grids. In Chapter 4, improved deferred schemes and additional pressure correction method are discussed. IDC shows less iterations steps compared with standard deferred correction scheme from our numerical experience. More obvious improvement comes from additional pressure correction steps. The results show that for skew grid cavity case, only solver with skewness correction

step can converge the flow compared with SIMPLE and SIMPLEC methods based on the same parameter settings.

Semi-coupled implicit solver, which couples the velocity components to construct block Jacobian, is proposed in this thesis with multi improved schemes and methods in order to stabilize the flow and adjoint solvers. Based on the velocity block Jacobian, the corresponding PSC modification is also derived and implemented.

The 2D cavity cases on series of regular and perturbed girds validates the improved flow solver. Fewer iteration numbers shows the better stability of semi-coupled solver compared with standard SIMPLE and SIMPLEC algorithms. And better grid quality shows more improvement. Quad mesh cases give more improvement of convergence from flow solvers compared with triangle ones since less iterations are needed. For perturbed grids, semi-coupled implicit with extra skewness correction needs more CPU time compared with SIMPLE and SIMPLEC solvers due to the additional computation of Poisson equation. 3D s-bend air duct cases at increasing Reynolds numbers give increasing challenges for flow and discrete solvers. From the comparison of convergence, semi-coupled implicit and SIMPLEC shows better stability compared with standard SIMPLE solver; for finer mesh, semi-coupled implicit solver with AMG linear solver for continuity equation uses the least CPU time and the lowest residual among all three test solvers.

In addition, the discrete adjoint solver with modified implementation not only converges all cases when primal converges, but also reduces nearly 50% CPU time compared with standard treatments. AMG linear solver shows improvement of CPU time as well.

Taking advantages of the improvement of flow and discrete adjoint solver, the gradient-based optimisation case study are conducted successfully for 3D s-bend air flow channel at different Reynolds numbers. CAD-based parametrisation termed 'NURBS-based parametrisation with continuity constraints' is applied to complete the shape optimisation design loop. In-house tool NsPCC is used for calculated the CAD derivative, and mesh deformation is based on linear elasticity analogy. For low Reynolds number, optimisation design loop doesn't require highly for flow and discrete solvers, the standard SIMPLE solver is applied safely. However, when Reynolds number increases, the flow becomes complex. The according complex shape change leads the SIMPLE solver blow-up and the interruption of optimisation. Instead, semi-coupled implicit solver is conducted for higher Reynolds number shape optimisation case.

Discrete adjoint sensitivity of spacers arranged as cavity, submerged and zigzag configuration in RO flow channel is the first time analysed in this study. Before simulating and optimising the spacer-filled RO membrane flow channel, the modified flow solver is validated with the experimental data and published data from litera-

tures. Using the discrete adjoint solver, sensitivity analysis is used to identify the mechanisms of the spacers affecting the pressure drop and permeation in different configurations. The gradient fields indicate that pressure drop gradients are similar among the different configurations. But the sensitive parts of spacer shape are varied among zigzag, submerged and cavity configuration. In addition, with respect to the pressure drop, a node-based approach optimisation is applied for spacers' shape in zigzag RO membrane flow channel. The optimised shape significantly reduces the pressure drop by 24% with the slight increase of permeation by 0.43%.

7.2 Future work

Although the robustness and stability of the incompressible flow solver is improved through various methods in this study, the case-dependency issue cannot be completely avoided. For each case, the flow solver is chosen and set specifically, so as we can find that the widely used CFD software (e.g. Fluent, CFX and OpenFOAM) contain a number of solvers, schemes and functionalities. This study unifies SIMPLE-like algorithms from the alternative PSC approach and combines different stabilisation strategies together in order to let the steady incompressible flow solver affordable for a wide range of cases and large parameters' work space. However, in the future, more strategies for improving stabilisation might need to be implemented when different types of physical flow problems are involved, e.g. optimisation problem in an unsteady flow environment.

Besides the flow and adjoint solvers, dealing with the gradients and re-meshing also significantly influence the optimisation loop. With the sensitivity, shape geometry is updated and an automatic mesh update or re-meshing should also be carried out for adapting the new shape. If there are not such tools, mesh deformation is usually used. Consequently, the performance of the mesh deformation algorithm significantly affects the convergence of the optimisation. If the quality of the mesh significantly deteriorates due to the accumulated deformation through a number of iterations, the optimisation will fail to converge. Therefore, strategies to improve the performance of the mesh deformation algorithm or reduce the mesh quality deterioration over optimisation iterations using re-meshing tools need to be studied.

Additionally, in the gradient-based optimisation loop using the discrete adjoint method, CFD simulations need to be running iteratively to obtain the optimum shape, which costs a large number of computations. Besides, in order to solver flow problems with fairly complex details or a large geometric computational domain, which requires highly-refined grids and the serial computation normally can not afford these cases. For the gradient-based optimisation in RO membrane process, further work will be carried out in to more realistic geometry of the SWM module. A

3D flow simulation will be developed to considering the curvature effect of the flow channel and the flows in the spiral direction. This requires more powerful CFD tools to deal with. To develop efficient flow and adjoint solvers, parallel computing is the further improvement, which simultaneous uses more than one processor to execute the codes. In flow based optimisation problems using adjoint method, parallelisation of the codes by domain decomposition requires to consider the codes implementation of flow solver, adjoint solver, mesh perturbation and gradient integration. Through the sensitivity analysis of the spacer shape with respect to different objective functions using adjoint method, mechanisms of spacers in the three dimension SWM module will be discussed.

Appendix A

GPDE Solver

Using object-oriented programming, GPDE possesses reusability, flexibility and extensibility. The major components includes:

- The *Base* module provides sorting and listing functions as basic components for efficient search; physical constants, like fluid physical properties: density and dynamic viscosity, and common mathematical constants are valued in this module;
- The *Mesh* module in GPDE reads and writes mesh files in the format of GMSH [1]; connectivity in this module gives graph of mesh elements' mapping and transforms mesh connectivity to cell-based data structure (mappings of faces and cells, boundary faces and abutting cells, faces and vertices and so on) in this module;
- *Geometry* module computes geometric quantities like face areas, volumes, normal vectors etc.; cell centroids, face center points and geometric weights for interpolation are calculated as well;
- The *Pdes* module undertakes the main component in GPDE for fluid flow computation, which builds up all discretilised partial differential equations, like generating the coefficient matrices and right hand sides of momentum equation, continuity equation and arbitrary scalar transportation equation;
- The *Utils* module contains all the basic mathematical functions (vectors' cross-product, dot-product and so on), fix-point control for the outer iterations, linear solvers for algebra equations and other miscellaneous subroutines (such as objective functions).

It needs to indicate that GPDE can provide non-dimensional and dimensional solutions. In non-dimensional computation, the density of fluid is constant and equals

[1] `http://geuz.org/gmsh/`

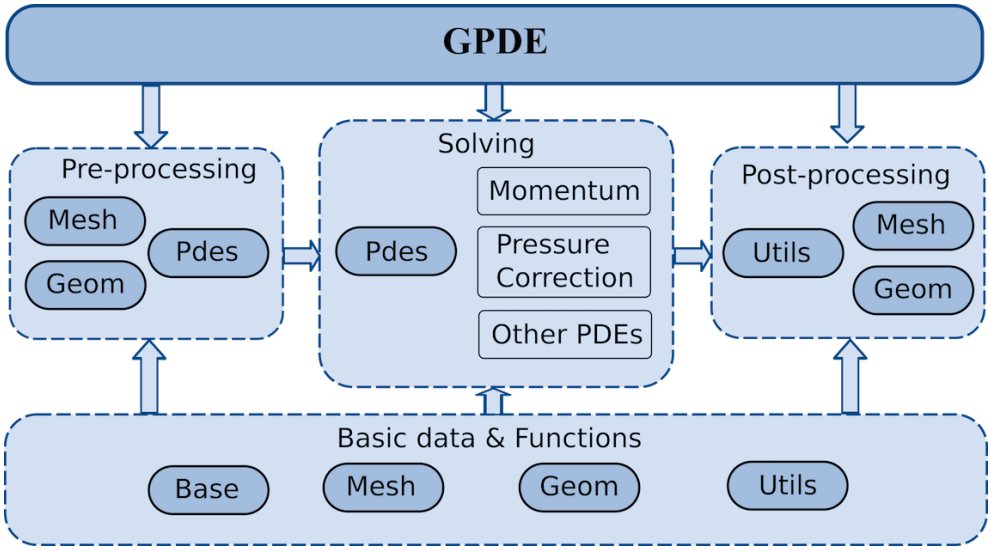


Figure A.1: Overview of GPDE structure

to one. Reference velocity equals to 1. Coefficient of convection and geometry characteristic length determines Reynolds number:

$$\begin{aligned} Re &= \frac{\rho U L}{\mu} \\ &= \frac{1 \times 1 \times L}{\Gamma} \end{aligned} \tag{A.1}$$

Based on similarity theory, Reynolds number is the key consideration besides geometric similarity for incompressible flow. For flow with low Mach number ($Ma < 0.3$), compressibility of fluid can be neglected, and steady incompressible NavierStokes momentum equation (without considering external force) can be simplified and non-dimentionalized as:

$$(\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \frac{1}{Re}\nabla^2 \mathbf{u} \tag{A.2}$$

where we can see only Reynolds number determines the flow. Flows with same Reynolds number shows the similar flow patten. For standard computation, all the variables adopt SI (standard international) units.

Appendix B

Performance of Algebraic Multi-grid Method

B.1 Given function case

Table B.1: Performance of BoomerAMG solver

Coarsen Type	Grid Number	Number of Level	Iteration Number	CPU Time	Node Runtime
CLJP(0)	10×10	5	5	3.3333×10^{-5}	3.333×10^{-7}
	50×50	8	10	3.1662×10^{-4}	1.266×10^{-7}
	100×100	10	11	1.4164×10^{-3}	1.416×10^{-7}
RS(3)	10×10	4	6	3.3333×10^{-5}	3.333×10^{-7}
	50×50	6	6	1.4998×10^{-4}	5.999×10^{-8}
	100×100	7	6	4.8326×10^{-4}	4.832×10^{-8}
	500×500	9	6	1.2848×10^{-2}	5.140×10^{-8}
Falgout(6)	10×10	4	6	1.6666×10^{-5}	1.667×10^{-7}
	50×50	6	6	1.8332×10^{-4}	7.332×10^{-8}
	100×100	7	6	5.4992×10^{-4}	5.499×10^{-8}
	500×500	9	6	1.3517×10^{-2}	5.407×10^{-8}
HMIS(10)	10×10	4	6	3.3316×10^{-5}	3.332×10^{-7}
	50×50	6	6	1.1665×10^{-4}	4.666×10^{-8}
	100×100	7	7	4.9992×10^{-4}	4.999×10^{-8}
	500×500	9	7	1.3748×10^{-2}	5.499×10^{-8}

Table B.2: Performance of BoomerAMG-pre PCG solver

Coarsen Type	Grid Number	Number of Level	Iteration Number	CPU Time	Node Runtime
CLJP(0)	10×10	5	4	3.3333×10^{-5}	3.333×10^{-7}
	50×50	8	6	3.1662×10^{-4}	1.266×10^{-7}
	100×100	10	6	1.1835×10^{-3}	1.184×10^{-7}
RS(3)	10×10	4	4	1.6666×10^{-5}	1.667×10^{-7}
	50×50	6	4	1.6663×10^{-4}	6.665×10^{-8}
	100×100	7	5	4.8326×10^{-4}	4.833×10^{-8}
	500×500	9	5	1.5814×10^{-2}	6.326×10^{-8}
Falgout(6)	10×10	4	4	1.6666×10^{-5}	1.667×10^{-7}
	50×50	6	4	1.4983×10^{-4}	5.993×10^{-8}
	100×100	7	5	5.3325×10^{-4}	5.333×10^{-8}
	500×500	9	5	1.3548×10^{-2}	5.419×10^{-8}
HMIS(10)	10×10	4	4	1.6666×10^{-5}	1.667×10^{-7}
	50×50	6	4	1.4998×10^{-4}	5.999×10^{-8}
	100×100	7	5	4.8325×10^{-4}	4.833×10^{-8}
	500×500	9	5	1.2898×10^{-2}	5.159×10^{-8}

B.2 2D lid-driven cavity case

Table B.3: Performance of BoomerAMG Solver with adaptive multi levels

Coarsen Type	Grid Number	Number of Level	Iteration Number	CPU Time	Node Runtime
CLJP(0)	50×50	8	4	1.1198×10^{-4}	4.479×10^{-8}
	100×100	10	3	3.9682×10^{-4}	3.968×10^{-8}
	500×500	13	3	1.0642×10^{-2}	4.257×10^{-8}
	1000×1000	14	2	4.1537×10^{-2}	4.154×10^{-8}
RS(3)	50×50	7	4	7.0983×10^{-5}	2.800×10^{-8}
	100×100	9	3	2.3118×10^{-4}	2.312×10^{-8}
	500×500	12	2	5.4510×10^{-3}	2.180×10^{-8}
	1000×1000	13	2	2.1850×10^{-2}	2.185×10^{-8}
Falgout(6)	50×50	7	4	7.3983×10^{-5}	2.958×10^{-8}
	100×100	9	3	2.5480×10^{-4}	2.548×10^{-8}
	500×500	12	2	5.7174×10^{-3}	2.286×10^{-8}
	1000×1000	14	2	2.3849×10^{-2}	2.385×10^{-8}
HMIS(10)	50×50	6	5	6.4967×10^{-5}	2.599×10^{-8}
	100×100	7	5	2.3227×10^{-4}	2.323×10^{-8}
	500×500	10	5	5.9361×10^{-3}	2.374×10^{-8}
	1000×1000	11	3	2.0467×10^{-2}	2.047×10^{-8}

Table B.4: Performance of BoomerAMG preconditioned PCG solver with adaptive multi levels

Coarsen Type	Grid Number	Number of Level	Iteration Number	CPU Time	Node Runtime
CLJP(0)	50×50	8	6	1.4883×10^{-4}	5.953×10^{-8}
	100×100	10	6	5.7498×10^{-4}	5.750×10^{-8}
	500×500	13	7	1.6672×10^{-2}	6.669×10^{-8}
	1000×1000	14	8	7.9689×10^{-2}	7.969×10^{-8}
RS(3)	50×50	7	5	9.2083×10^{-5}	3.683×10^{-8}
	100×100	9	6	3.3903×10^{-4}	3.390×10^{-8}
	500×500	12	7	9.7822×10^{-3}	3.913×10^{-8}
	1000×1000	13	19	7.6555×10^{-2}	7.656×10^{-8}
Falgout(6)	50×50	7	5	9.9683×10^{-5}	3.072×10^{-8}
	100×100	9	6	3.5313×10^{-4}	3.531×10^{-8}
	500×500	12	6	9.3074×10^{-3}	3.723×10^{-8}
	1000×1000	14	7	4.2258×10^{-2}	4.226×10^{-8}
HMIS(10)	50×50	6	7	8.4550×10^{-5}	3.382×10^{-8}
	100×100	7	8	3.2932×10^{-4}	3.293×10^{-8}
	500×500	10	11	1.0394×10^{-2}	4.158×10^{-8}
	1000×1000	11	13	4.9103×10^{-2}	4.910×10^{-8}

Table B.5: Performance of BoomerAMG solver with 2 levels

Coarsen Type	Grid Number	Iteration Number	CPU Time	Node Runtime
CLJP(0)	50×50	200	1.4998×10^{-3}	5.999×10^{-7}
	100×100	510	1.4981×10^{-2}	1.488×10^{-6}
	500×500	1000+	9.4180×10^{-1}	9.418×10^{-5}
	1000×1000	371	1.4747×10^{0}	1.475×10^{-6}
RS(3)	50×50	171	1.0332×10^{-3}	4.133×10^{-7}
	100×100	433	1.0465×10^{-2}	1.047×10^{-6}
	500×500	1000+	8.7870×10^{-1}	3.515×10^{-6}
	1000×1000	309	1.1391×10^{0}	1.139×10^{-6}
Falgout(6)	50×50	171	1.0498×10^{-3}	4.186×10^{-7}
	100×100	433	1.0448×10^{-2}	1.045×10^{-6}
	500×500	1000+	8.3960×10^{-1}	3.358×10^{-6}
	1000×1000	309	1.1580×10^{0}	1.158×10^{-6}
HMIS(10)	50×50	171	1.0498×10^{-3}	4.200×10^{-7}
	100×100	433	1.0465×10^{-2}	1.046×10^{-6}
	500×500	1000+	8.4222×10^{-1}	3.369×10^{-6}
	1000×1000	309	1.1051×10^{0}	1.105×10^{-6}

Table B.6: Performance of BoomerAMG preconditioned PCG solver with 2 levels

Coarsen Type	Grid Number	Iteration Number	CPU Time	Node Runtime
CLJP(0)	50×50	43	5.9992×10^{-4}	2.400×10^{-7}
	100×100	78	4.1160×10^{-3}	4.116×10^{-7}
	500×500	373	1.3383×10^{0}	5.353×10^{-6}
	1000×1000	677	8.6255×10^{0}	8.626×10^{-6}
RS(3)	50×50	39	4.1662×10^{-4}	1.667×10^{-7}
	100×100	64	2.5496×10^{-3}	2.550×10^{-7}
	500×500	310	1.1874×10^{0}	4.750×10^{-6}
	1000×1000	615	6.4530×10^{0}	6.453×10^{-6}
Falgout(6)	50×50	39	4.3327×10^{-4}	1.733×10^{-7}
	100×100	64	2.5496×10^{-3}	2.550×10^{-7}
	500×500	310	6.2650×10^{0}	2.506×10^{-5}
	1000×1000	615	5.1921×10^{0}	5.192×10^{-6}
HMIS(10)	50×50	39	4.1660×10^{-4}	1.666×10^{-7}
	100×100	64	2.3496×10^{-3}	2.350×10^{-7}
	500×500	310	6.1010×10^{0}	2.440×10^{-5}
	1000×1000	615	5.2415×10^{0}	5.242×10^{-6}

Table B.7: Performance of BoomerAMG solver with 5 levels

Coarsen Type	Grid Number	Iteration Number	CPU Time	Node Runtime
CLJP(0)	50×50	13	3.1662×10^{-4}	1.266×10^{-7}
	100×100	32	2.4329×10^{-3}	2.433×10^{-7}
	500×500	160	3.1950×10^{-1}	2.178×10^{-6}
	1000×1000	26	2.7080×10^{-1}	2.708×10^{-7}
RS(3)	50×50	7	1.4997×10^{-4}	5.999×10^{-8}
	100×100	15	9.6652×10^{-4}	9.665×10^{-8}
	500×500	82	1.2150×10^{-1}	4.860×10^{-7}
	1000×1000	8	7.7838×10^{-2}	7.784×10^{-8}
Falgout(6)	50×50	7	1.6663×10^{-4}	6.665×10^{-8}
	100×100	15	9.9985×10^{-4}	9.999×10^{-8}
	500×500	82	1.2015×10^{-1}	4.806×10^{-7}
	1000×1000	8	8.4104×10^{-2}	8.410×10^{-8}
HMIS(10)	50×50	6	1.4998×10^{-4}	5.999×10^{-8}
	100×100	10	6.6657×10^{-4}	6.666×10^{-8}
	500×500	29	4.4892×10^{-2}	1.796×10^{-7}
	1000×1000	5	5.6308×10^{-2}	5.631×10^{-8}

Table B.8: Performance of BoomerAMG preconditioned PCG solver with 5 levels

Coarsen Type	Grid Number	Iteration Number	CPU Time	Node Runtime
CLJP(0)	50×50	11	4.8327×10^{-4}	1.933×10^{-7}
	100×100	20	2.8162×10^{-3}	2.816×10^{-7}
	500×500	97	3.4960×10^{-1}	1.398×10^{-6}
	1000×1000	178	2.6101×10^{0}	2.610×10^{-6}
RS(3)	50×50	8	2.8330×10^{-4}	9.133×10^{-8}
	100×100	13	1.4331×10^{-3}	1.433×10^{-7}
	500×500	66	1.7030×10^{-1}	6.810×10^{-7}
	1000×1000	129	1.3772×10^{0}	1.377×10^{-6}
Falgout(6)	50×50	8	2.8328×10^{-4}	1.133×10^{-7}
	100×100	13	1.4664×10^{-3}	1.466×10^{-7}
	500×500	66	1.7261×10^{-1}	6.900×10^{-7}
	1000×1000	129	1.3412×10^{0}	1.341×10^{-6}
HMIS(10)	50×50	8	2.4996×10^{-4}	1.000×10^{-7}
	100×100	12	1.2331×10^{-3}	1.233×10^{-7}
	500×500	48	1.1570×10^{-1}	4.630×10^{-7}
	1000×1000	94	9.0310×10^{-1}	9.031×10^{-7}

Table B.9: Performance of BoomerAMG solver with 10 levels

Coarsen Type	Grid Number	Iteration Number	CPU Time	Node Runtime
CLJP(0)	50×50/8 level	4	1.9997×10^{-4}	8.000×10^{-8}
	100×100	3	7.1657×10^{-4}	7.166×10^{-8}
	500×500	4	2.3729×10^{-2}	9.490×10^{-8}
	1000×1000	2	8.0988×10^{-2}	8.099×10^{-8}
RS(3)	50×50/7 level	4	1.4998×10^{-4}	6.000×10^{-8}
	100×100/9 level	3	4.4993×10^{-4}	4.499×10^{-8}
	500×500	2	1.1798×10^{-2}	4.720×10^{-8}
	1000×1000	2	4.8343×10^{-2}	4.834×10^{-8}
Falgout(6)	50×50/7 level	4	1.3332×10^{-4}	5.330×10^{-8}
	100×100/9 level	3	4.8327×10^{-4}	4.833×10^{-8}
	500×500	2	1.2165×10^{-2}	4.870×10^{-8}
	1000×1000	2	5.4625×10^{-2}	5.463×10^{-8}
HMIS(10)	50×50/6 level	5	1.1663×10^{-4}	4.670×10^{-8}
	100×100/7 level	5	4.9993×10^{-4}	4.999×10^{-8}
	500×500	5	1.4731×10^{-2}	5.890×10^{-8}
	1000×1000	3	4.8826×10^{-2}	4.883×10^{-8}

Table B.10: Performance of BoomerAMG preconditioned PCG solver with 10 levels

Coarsen Type	Grid Number	Iteration Number	CPU Time	Node Runtime
CLJP(0)	$50 \times 50/8$ level	6	2.6662×10^{-4}	1.067×10^{-7}
	100×100	6	1.1498×10^{-3}	1.150×10^{-7}
	500×500	13	6.9973×10^{-2}	2.799×10^{-7}
	1000×1000	23	4.4490×10^{-1}	4.449×10^{-7}
RS(3)	$50 \times 50/7$ level	5	1.8332×10^{-4}	7.333×10^{-8}
	$100 \times 100/9$ level	6	8.3320×10^{-4}	8.332×10^{-8}
	500×500	8	3.4845×10^{-2}	1.394×10^{-7}
	1000×1000	13	2.0111×10^{-1}	2.011×10^{-7}
Falgout(6)	$50 \times 50/7$ level	5	1.8329×10^{-4}	7.330×10^{-8}
	$100 \times 100/9$ level	6	8.3322×10^{-4}	8.332×10^{-8}
	500×500	8	3.6427×10^{-2}	1.457×10^{-7}
	1000×1000	13	2.0842×10^{-1}	2.084×10^{-7}
HMIS(10)	$50 \times 50/6$ level	7	1.8330×10^{-4}	7.330×10^{-8}
	$100 \times 100/7$ level	8	8.6653×10^{-4}	8.665×10^{-8}
	500×500	11	3.8261×10^{-2}	1.530×10^{-7}
	1000×1000	13	1.7910×10^{-1}	1.791×10^{-7}

B.3 3D S-bend air channel case

Table B.11: Performance of BoomerAMG solver with adaptive multi levels

Coarsen Type	Grid Number	Number of Level	Iteration Number	CPU Time	Node Runtime
CLJP(0)	$47k$	11	15	1.7880×10^{-2}	3.800×10^{-7}
	$130k$	12	19	6.2257×10^{-2}	4.790×10^{-7}
	$500k$	14	26	4.6580×10^{-1}	9.300×10^{-7}
RS(3)	$47k$	10	NC[1]	NC	NC
	$130k$	11	20	4.9076×10^{-2}	3.770×10^{-7}
	$500k$	12	26	2.9998×10^{-1}	6.000×10^{-7}
Falgout(6)	$47k$	10	NC	NC	NC
	$130k$	11	NC	NC	NC
	$500k$	12	25	3.0890×10^{-1}	6.200×10^{-7}
HMIS(10)	$47k$	7	388	9.6852×10^{-2}	2.061×10^{-6}
	130	8	46	4.1694×10^{-2}	3.210×10^{-7}
	$500k$	8	98	3.5980×10^{-1}	7.200×10^{-7}

All test cases diverge when using BoomerAMG Solver with 2 and 5 levels' grids.

[1]Not Converge

Table B.12: Performance of BoomerAMG preconditioned PCG solver with adaptive multi levels

Coarsen Type	Grid Number	Number of Level	Iteration Number	CPU Time	Node Runtime
CLJP(0)	$47k$	11	7	1.1198×10^{-4}	2.380×10^{-9}
	$130k$	12	8	3.9682×10^{-4}	3.050×10^{-9}
	$500k$	14	9	1.0642×10^{-2}	2.100×10^{-8}
RS(3)	$47k$	10	67	7.0983×10^{-5}	1.510×10^{-9}
	$130k$	11	8	2.3118×10^{-4}	1.780×10^{-9}
	$500k$	12	9	5.4510×10^{-3}	1.090×10^{-8}
Falgout(6)	$47k$	10	67	7.3983×10^{-5}	1.574×10^{-9}
	$130k$	11	1000+	2.5480×10^{-4}	1.960×10^{-9}
	$500k$	12	9	5.7174×10^{-3}	1.140×10^{-8}
HMIS(10)	$47k$	7	12	6.4967×10^{-5}	1.382×10^{-9}
	130	8	14	2.3227×10^{-4}	1.790×10^{-9}
	$500k$	8	17	5.9361×10^{-3}	1.190×10^{-8}

Table B.13: Performance of BoomerAMG preconditioned PCG solver with 2 levels

Coarsen Type	Grid Number	Iteration Number	CPU Time	Node Runtime
CLJP(0)	$47k$	99	1.7514×10^{-2}	3.730×10^{-7}
	$130k$	144	7.8705×10^{-2}	6.050×10^{-7}
	$500k$	228	5.3660×10^{0}	1.070×10^{-5}
RS(3)	$47k$	94	1.6964×10^{-2}	3.610×10^{-7}
	$130k$	136	6.8606×10^{-2}	5.280×10^{-7}
	$500k$	213	4.6270×10^{0}	9.300×10^{-6}
Falgout(6)	$47k$	94	1.4881×10^{-2}	3.170×10^{-7}
	$130k$	136	6.8689×10^{-2}	5.280×10^{-7}
	$500k$	213	4.6219×10^{0}	9.200×10^{-6}
HMIS(10)	$47k$	94	1.4314×10^{-2}	3.050×10^{-7}
	130	135	6.6906×10^{-2}	5.150×10^{-7}
	$500k$	214	4.6218×10^{0}	9.200×10^{-6}

Table B.14: Performance of BoomerAMG preconditioned PCG solver with 5 levels

Coarsen Type	Grid Number	Iteration Number	CPU Time	Node Runtime
CLJP(0)	$47k$	30	2.8712×10^{-2}	6.110×10^{-7}
	$130k$	42	1.1111×10^{0}	8.500×10^{-6}
	$500k$	70	8.6013×10^{0}	1.720×10^{-5}
RS(3)	$47k$	24	1.8980×10^{-2}	4.040×10^{-7}
	$130k$	36	7.8121×10^{-2}	6.010×10^{-7}
	$500k$	53	5.0341×10^{0}	1.010×10^{-5}
Falgout(6)	$47k$	24	1.5114×10^{-2}	3.220×10^{-7}
	$130k$	36	7.8671×10^{-2}	6.050×10^{-7}
	$500k$	53	5.0701×10^{0}	1.010×10^{-5}
HMIS(10)	$47k$	20	7.6822×10^{-2}	1.635×10^{-6}
	130	28	3.6894×10^{-2}	2.840×10^{-7}
	$500k$	40	2.4476×10^{0}	4.900×10^{-6}

Table B.15: Performance of BoomerAMG solver with 10 levels

Coarsen Type	Grid Number	Iteration Number	CPU Time	Node Runtime
CLJP(0)	$47k$	652	1.9540×10^{1}	4.157×10^{-4}
	$130k$	155	8.4340×10^{-1}	6.490×10^{-6}
	$500k$	652	2.2737×10^{1}	4.840×10^{-4}
RS(3)	$47k$	NC	NC	NC
	$130k$	52	2.0237×10^{-1}	1.560×10^{-6}
	$500k$	279	5.4211×10^{0}	1.080×10^{-5}
Falgout(6)	$47k$	NC	NC	NC
	$130k$	67	2.5190×10^{-1}	1.940×10^{-6}
	$500k$	284	5.3540×10^{0}	1.070×10^{-5}
HMIS(10)	$47k$	388	9.6985×10^{-2}	2.064×10^{-6}
	$130k/8$ level	46	9.3902×10^{-2}	7.220×10^{-7}
	$500k/8$ level	98	1.1658×10^{0}	2.300×10^{-6}

Table B.16: Performance of BoomerAMG preconditioned PCG solver with 10 levels

Coarsen Type	Grid Number	Iteration Number	CPU Time	Node Runtime
CLJP(0)	$47k$	8	1.6931×10^{-2}	3.600×10^{-7}
	$130k$	11	6.2207×10^{-2}	4.790×10^{-7}
	$500k$	17	4.8694×10^{0}	9.700×10^{-6}
RS(3)	$47k$	67	5.4741×10^{-2}	1.165×10^{-6}
	$130k$	9	4.2843×10^{-2}	3.296×10^{-7}
	$500k$	12	2.5839×10^{0}	5.168×10^{-6}
Falgout(6)	$47k$	67	5.5108×10^{-2}	1.172×10^{-6}
	$130k$	9	4.3210×10^{-2}	3.324×10^{-7}
	$500k$	12	2.6154×10^{0}	5.230×10^{-6}
HMIS(10)	$47k$	12	5.4658×10^{-2}	1.163×10^{-6}
	$130k/8$ level	14	2.6612×10^{-2}	2.047×10^{-7}
	$500k/8$ level	17	1.3609×10^{0}	2.722×10^{-6}

Bibliography

[1] J-D Müller, M Jitsumura, and NHF Müller-Kronast. Sensitivity of flow simulations in a cerebral aneurysm. *Journal of biomechanics*, 45(15):2539–2548, 2012.

[2] Bongjae Chung and Juan Raul Cebral. Cfd for evaluation and treatment planning of aneurysms: review of proposed clinical uses and their challenges. *Annals of biomedical engineering*, 43(1):122–138, 2015.

[3] ANSYS Fluent group. Fluent. `http://www.ansys.com/Products/Fluids/ANSYS-Fluent`, 2016. [Online; accessed 9-May-2016].

[4] ANSYS CFX group. Cfx. `http://www.ansys.com/Products/Fluids/ANSYS-CFX`, 2016. [Online; accessed 9-May-2016].

[5] STAR-CD group. CD-adapco STAR-CD. `http://www.cd-adapco.com/products/star-cd`, 2016. [Online; accessed 9-May-2016].

[6] OpenFOAM group. OpenFOAM. `http://openfoam.org`, 2016. [Online; accessed 9-May-2016].

[7] SU2 group. SU2. `http://su2.stanford.edu`, 2016. [Online; accessed 9-May-2016].

[8] Dolfyn group. The Open Source CFD code Dolfyn. `http://www.dolfyn.net/dolfyn/index_en.html`, 2016. [Online; accessed 9-May-2016].

[9] A. Thom. The flow past circular cylinder at low speeds. In *Proceedings of Royal Society of London*, pages 652–666, 1993.

[10] A. Jameson, L. Martinelli, and N.A. Pierce. Optimum aerodynamic design using the Navier-Stokes equations. *Theor. Comp. Fluid. Dyn.*, 10:213–237, 1998.

[11] M. B. Giles, M. C. Duta, J.-D Müller, and N. A. Pierce. Algorithm developments for discrete adjoint methods. *AIAA Journal*, 41(2):198–205, 2003.

[12] S. Kammerer, J.F., M. Paffrath, U. Wever, and A.R. Jung. Three-dimensional optimization of turbomachinery bladings using sensitivity analysis. *AIAA paper*, (GT2003-38037), 2003.

[13] H.D. Li, L. He, Y.S. Li, and R. Wells. Blading aerodynamics design optimization with mechanical and aeromechanical constraints. *ASME Paper*, (GT2006-90503), 2006.

[14] C. Othmer and T. Grahs. Approaches to fluid dynamic optimization in the car development process. In R. Schilling et. al., editor, *Eurogen*, Munich, 2005. Eccomas.

[15] C. Othmer, T. Kaminski, and R. Giering. Computation of topological sensitivities in fluid dynamics: cost function versatility. In *Eccomas CFD 2006*. Eccomas, P. Wesseling and E. Oñate and J. Périaux, 2006.

[16] David W Zingg, Marian Nemec, and Thomas H Pulliam. A comparative evaluation of genetic and gradient-based algorithms applied to aerodynamic optimization. *European Journal of Computational Mechanics/Revue Européenne de Mécanique Numérique*, 17(1-2):103–126, 2008.

[17] A. Jameson. Optimum aerodynamic design using control theory. In M. Hafez and K. Oshima, editors, *Computational Fluid Dynamics Review 1995*, pages 492–528. John Wiley & Sons, 1995.

[18] Antony Jameson and Luigi Martinelli. Aerodynamic shape optimization techniques based on control theory. In *Computational Mathematics Driven by Industrial Problems*, pages 151–221. Springer, 2000.

[19] Antony Jameson. Aerodynamic shape optimization using the adjoint method. *Lectures at the Von Karman Institute, Brussels*, 2003.

[20] Grégoire Allaire. A review of adjoint methods for sensitivity analysis, uncertainty quantification and optimization in numerical codes. *Ingénieurs de l'Automobile*, 836:33–36, 2015.

[21] Shenren Xu. *CAD-based CFD shape optimisation using discrete adjoint solvers*. PhD thesis, Queen Mary University of London, 2015.

[22] Shenren Xu, Wolfram Jahn, and Jens-Dominik Müller. Cad-based shape optimisation with cfd using a discrete adjoint. *International Journal for Numerical Methods in Fluids*, 74(3):153–168, 2014.

[23] H Bijl and Pieter Wesseling. A numerical method for the computation of compressible flows with low mach number regions. In *ECCOMAS computational fluid dynamics conference*, volume 206. JOHN WILEY & SONS Limited, 1996.

[24] Stefan Turek. *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approache*, volume 6. Springer Science & Business Media, 1999.

[25] Y-H Choi and Charles L Merkle. The application of preconditioning in viscous flows. *Journal of Computational Physics*, 105(2):207–223, 1993.

[26] IJ Keshtiban, F Belblidia, and MF Webster. Compressible flow solvers for low mach number flowsa review. *Int. J. Numer. Methods Fluids*, 23:77–103, 2004.

[27] C-D Munz, Sabine Roller, Rupert Klein, and Karl J Geratz. The extension of incompressible flow solvers to the weakly compressible regime. *Computers & Fluids*, 32(2):173–196, 2003.

[28] A. Chorin. A numerical method for solving incompressible viscous flow problems. *Journal of Computational Physics*, 2(1):12–26, 1967.

[29] Alexandre Joel Chorin. Numerical solution of the navier-stokes equations. *Mathematics of computation*, 22(104):745–762, 1968.

[30] Suhas V Patankar. A calculation procedure for two-dimensional elliptic situations. *Numerical heat transfer*, 4(4):409–425, 1981.

[31] S. V. Patankar and D.B Spalding. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flow. *Int. J. of Heat and Mass Transfer*, 15(10):1787–1806, 1972.

[32] S. V. Patankar. A calculation procedure for two-dimensional elliptic situations. *Int. J. of Heat and Mass Transfer*, 4:409–425, 1981.

[33] J. P. Van Doormal and G. D Raithby. Enhancements of the simple method for predicting incompressible fluid flows. *Numerical Heat Transfer*, 7(2):147–63, 1984.

[34] M Perić, R Kessler, and G Scheuerer. Comparison of finite-volume numerical methods with staggered and colocated grids. *Computers & Fluids*, 16(4):389–403, 1988.

[35] CM Rhie and WL Chow. Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA journal*, 21(11):1525–1532, 1983.

[36] Maxim A Olshanskii and Eugene E Tyrtshnikov. *Iterative methods for linear systems: theory and applications*. SIAM, 2014.

[37] Andy Wathen. Preconditioning and convergence in the right norm. *International Journal of Computer Mathematics*, 84(8):1199–1209, 2007.

[38] Jörg Liesen and Zdenek Strakos. *Krylov subspace methods: principles and analysis*. OUP Oxford, 2012.

[39] Michele Benzi. Preconditioning techniques for large linear systems: a survey. *Journal of computational Physics*, 182(2):418–477, 2002.

[40] Jacques-Louis Lions. *Optimal control of systems governed by partial differential equations*. Springer, 1971.

[41] A. Jameson. Optimum aerodynamic design using CFD and control theory. *AIAA paper*, 1729:124–131, 1995.

[42] O. Pironneau. On optimum design in fluid mechanics. *Journal of Fluid Mechanics*, 64:97–110, 1974.

[43] A. Jameson. Aerodynamic design via control theory. *Journal of Scientific Computing*, 3:233–260, 1988.

[44] James Reuther, Antony Jameson, James Farmer, Luigi Martinelli, and David Saunders. Aerodynamic shape optimization of complex aircraft configurations via an adjoint formulation. *AIAA paper*, 94, 1996.

[45] J. Reuther, A. Jameson, J.J. Alonso, and M.J. Remlinger. Constrained multipont aerodynamic shape optimisation using an adjoint formulation and parallel computers, part 1. *Journal of Aircraft*, 36(1):51–60, 1999.

[46] J. Reuther, A. Jameson, J.J. Alonso, and M.J. Remlinger. Constrained multipont aerodynamic shape optimisation using an adjoint formulation and parallel computers, part 2. *Journal of Aircraft*, 36(1):61–74, 1999.

[47] J. Reuther, J.J. Alonso, M.J. Rimlinger, and A. Jameson. Aerodynamic shape optimization of supersonic aircraft configurations via an adjoint formulation on distributed memory parallel computers. *Computers and Fluids*, 28:675700, 1999.

[48] James Reuther and Antony Jameson. Supersonic wing and wing-body shape optimization using an adjoint formulation. 1995.

[49] Sangho Kim, Juan J Alonso, and Antony Jameson. A gradient accuracy study for the adjoint-based navier-stokes design method. *AIAA paper*, 299, 1999.

[50] Antony Jameson. Computational aerodynamics for aircraft design. *Science(Washington)*, 245(4916):361–371, 1989.

[51] Antony Jameson. Automatic design of transonic airfoils to reduce the shock induced pressure drag. In *Proceedings of the 31st Israel annual conference on aviation and aeronautics, Tel Aviv*, pages 5–17. Citeseer, 1990.

[52] Carsten Othmer. Adjoint methods for car aerodynamics. *Journal of Mathematics in Industry*, 4(1):1–23, 2014.

[53] Carsten Othmer, Eugene de Villiers, and Henry G Weller. Implementation of a continuous adjoint for topology optimization of ducted flows. In *18th AIAA Computational Fluid Dynamics Conference, June*, 2007.

[54] Gilles Marck, Maroun Nemer, and Jean-Luc Harion. Topology optimization of heat and mass transfer problems: laminar flow. *Numerical Heat Transfer, Part B: Fundamentals*, 63(6):508–539, 2013.

[55] Tijs Van Oevelen and Martine Baelmans. Numerical topology optimization of heat sinks. In *Proceedings of the 15th International Heat Transfer Conference*, pages 10–15. Begell House Publishers, 2014.

[56] EA Kontoleontos, EM Papoutsis-Kiachagias, AS Zymaris, DI Papadimitriou, and KC Giannakoglou. Adjoint-based constrained topology optimization for viscous flows, including heat transfer. *Engineering Optimization*, 45(8):941–961, 2013.

[57] Wenjing Yan and Zhiming Gao. Shape optimization in the navier-stokes flow with thermal effects. *Numerical Methods for Partial Differential Equations*, 30(5):1700–1715, 2014.

[58] Joe Alexandersen, Niels Aage, Casper Schousboe Andreasen, and Ole Sigmund. Topology optimisation for natural convection problems. *International Journal for Numerical Methods in Fluids*, 76(10):699–721, 2014.

[59] Simon W Funke, Patrick E Farrell, and MD Piggott. Tidal turbine array optimisation using the adjoint approach. *Renewable Energy*, 63:658–673, 2014.

[60] DM Culley, SW Funke, SC Kramer, and MD Piggott. Integration of cost modelling within the micro-siting design optimisation of tidal turbine arrays. *Renewable Energy*, 85:215–227, 2016.

[61] M. B. Giles and N. A. Pierce. An introduction to the adjoint approach to design. *Flow, Turb. Comb.*, 65:393–415, 2000. also Oxford University Computing Laboratory NA report 00/04.

[62] S. Nadarajah and A. Jameson. A comparison of the continuous and discrete adjoint approach to automatic aerodynamic optimization. *AIAA CP-00-0667*, 2000.

[63] J. Peter and R. Dwight. Numerical sensitivity analysis for aerodynamic optimization: A survey of approaches. *Computers and Fluids*, 39(3):373–391, 2010.

[64] H Martin Bücker, George Corliss, Paul Hovland, Uwe Naumann, and Boyana Norris. *Automatic differentiation: applications, theory, and implementations*, volume 50. Springer Science & Business Media, 2006.

[65] Jean Utke, Uwe Naumann, Mike Fagan, Nathan Tallent, Michelle Strout, Patrick Heimbach, Chris Hill, and Carl Wunsch. Openad/f: A modular open-source tool for automatic differentiation of fortran codes. *ACM Transactions on Mathematical Software (TOMS)*, 34(4):18, 2008.

[66] Andrea Walther, Andreas Griewank, and Olaf Vogel. Adol-c: Automatic differentiation using operator overloading in c++. *PAMM*, 2(1):41–44, 2003.

[67] Laurent Hascoet and Valérie Pascual. The tapenade automatic differentiation tool: principles, model, and specification. *ACM Transactions on Mathematical Software (TOMS)*, 39(3):20, 2013.

[68] J. Schwinge, D.E. Wiley, and A.G. Fane. Novel spacer design improves observed flux. *Journal of Membrane Science*, 229(12):53 – 61, 2004.

[69] SS Sablani, MFA Goosen, R Al-Belushi, and M Wilf. Concentration polarization in ultrafiltration and reverse osmosis: a critical review. *Desalination*, 141(3):269 – 289, 2001.

[70] C. Fritzmann, J. Lwenberg, T. Wintgens, and T. Melin. State-of-the-art of reverse osmosis desalination. *Desalination*, 216(13):1 – 76, 2007.

[71] Vitor Geraldes, Viriato Semio, and Maria Norberta Pinho. Hydrodynamics and concentration polarization in nf/ro spiral-wound modules with ladder-type spacers. *Desalination*, 157(13):395 – 402, 2003. Desalination and the Environment: Fresh Water for all.

[72] Vtor Geraldes, Viriato Semio, and Maria Norberta de Pinho. The effect of the ladder-type spacers configuration in {NF} spiral-wound modules on the concentration boundary layers disruption. *Desalination*, 146(13):187 – 194, 2002.

[73] Vtor Geraldes, Viriato Semio, and Maria Norberta de Pinho. Flow management in nanofiltration spiral wound modules with ladder-type spacers. *Journal of Membrane Science*, 203(12):87 – 102, 2002.

[74] J.L.C. Santos, V. Geraldes, S. Velizarov, and J.G. Crespo. Investigation of flow patterns and mass transfer in membrane module channels filled with flow-aligned spacers using computational fluid dynamics (cfd). *Journal of Membrane Science*, 305(12):103 – 117, 2007.

[75] S. Wardeh and H.P. Morvan. CFD simulations of flow and concentration polarization in spacer-filled channels for application to water desalination. *Chemical Engineering Research and Design*, 86(10):1107 – 1116, 2008.

[76] A.L. Ahmad, K.K. Lau, and M.Z. Abu Bakar. Impact of different spacer filament geometries on concentration polarization control in narrow membrane channel. *Journal of Membrane Science*, 262(12):138 – 152, 2005.

[77] M. Shakaib, S.M.F. Hasani, and M. Mahmood. {CFD} modeling for flow and mass transfer in spacer-obstructed membrane feed channels. *Journal of Membrane Science*, 326(2):270 – 284, 2009.

[78] Jian-Yuan Lee, Wen See Tan, Jia An, Chee Kai Chua, Chuyang Y. Tang, Anthony G. Fane, and Tzyy Haur Chong. The potential to enhance membrane module design with 3d printing technology. *Journal of Membrane Science*, 499:480 – 490, 2016.

[79] Asim Saeed, Rupa Vuthaluru, and Hari B. Vuthaluru. Impact of feed spacer filament spacing on mass transport and fouling propensities of ro membrane surfaces. *Chemical Engineering Communications*, 202(5):634–646, 2015.

[80] Greg Guillen and Eric M.V. Hoek. Modeling the impacts of feed spacer geometry on reverse osmosis and nanofiltration processes. *Chemical Engineering Journal*, 149(13):221 – 231, 2009.

[81] Mounir Amokrane, Djamel Sadaoui, Michel Dudeck, and Chrysafenia P. Koutsou. New spacer designs for the performance improvement of the zigzag spacer configuration in spiral-wound membrane modules. *Desalination and Water Treatment*, 57(12):5266–5274, 2016.

[82] Dhananjay Dendukuri, Sandeep K. Karode, and Ashwani Kumar. Flow visualization through spacer filled channels by computational fluid dynamics-ii: improved feed spacer designs. *Journal of Membrane Science*, 249(12):41 – 49, 2005.

[83] Vivek V. Ranade and Ashwani Kumar. Fluid dynamics of spacer filled rectangular and curvilinear channels. *Journal of Membrane Science*, 271(12):1 – 15, 2006.

[84] Jianxin Liu, Zhijun Liu, Fengxia Liu, Wei Wei, Zhiyi Li, Xiaojuan Wang, Chao Dong, and Xiaofei Xu. The application of saw-tooth promoter for flat sheet membrane filtration. *Desalination*, 359:149 – 155, 2015.

[85] Jianxin Liu, Zhijun Liu, Xiaofei Xu, and Fengxia Liu. Saw-tooth spacer for membrane filtration: Hydrodynamic investigation by {PIV} and filtration experiment validation. *Chemical Engineering and Processing: Process Intensification*, 91:23 – 34, 2015.

[86] Speed of sound in air at temperatures from -40 - 1000oc (-40 - 1500of) at standard atmospheric pressure - imperial and si units. `http://www.engineeringtoolbox.com/air-speed-sound-d_603.html`, 2016. [Online; accessed 27-May-2016].

[87] Speed of sound in water at temperatures ranging 32 - 212of (0 - 100oc) - imperial and si units. `http://www.engineeringtoolbox.com/sound-speed-water-d_598.html`, 2016. [Online; accessed 27-May-2016].

[88] S.V. Patankar and D.B. Spalding. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. *International Journal for Heat Mass Transfer*, 15:1787–1806, 1972.

[89] James Beverly Jones and Regina E Dugan. *Engineering thermodynamics*. Prentice Hall, 1996.

[90] Henk Kaarle Versteeg and Weeratunge Malalasekera. *An introduction to computational fluid dynamics: the finite volume method*. Pearson Education, 2007.

[91] J. Blazek. *Computational fluid dynamics : principles and applications*. Elsevier, 2005.

[92] Charles L Lawson and Richard J Hanson. *Solving least squares problems*, volume 15. SIAM, 1995.

[93] J.H. Ferziger and M. Perić. *Computational Methods for Fluid Dynamics*. Springer, 2002.

[94] Henk A Van der Vorst. Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM Journal on scientific and Statistical Computing*, 13(2):631–644, 1992.

[95] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[96] U. Ghia, K.N. Ghia, and C.T. Shin. High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method. *Journal of Computational Physics*, 48:387–411, 1982.

[97] Jens-Dominik Müller. A review of adjoint method. Technical report, Queen Mary, University of London, 2007.

[98] P. Cusdin. *Automatic sensitivity code for Computational Fluid Dynamics*. PhD thesis, School of Aeronautical Engineering, Queen's University Belfast, 2005.

[99] C. Othmer. A continuous adjoint formulation for the computation of topological and surface sensitivities of ducted flows. *International Journal for Numerical Methods in Fluids*, 58(8):861, 2008. DOI 10.1002/fld.1770.

[100] L. Hascoët. Tapenade: a tool for automatic differentiation of programs. In *Proceedings of 4th European Congress on Computational Methods, ECCOMAS*, 2004.

[101] B. Christianson. Reverse accumulation and implict functions. *Optimization Methods and Software*, 9(4):307–322, 1998.

[102] D. Jones, Jens-Dominik Müller, and F. Christakopoulos. Preparation and assembly of adjoint CFD codes. *Computers and Fluids*, 46(1), 2011.

[103] M.B. Giles, D. Ghate, and M.C. Duta. Using automatic differentiation for adjoint cfd code development. In B. Uthup, S.-P. Koruthu, R.-K. Sharma, and P. Priyadarshi, editors, *Recent Trends in Aerospace Design and Optimization*. Tata-McGraw Hill, New Delhi, 2005. Post-SAROD-2005, Bangalore, India.

[104] The developers of TAPENADE. The TAPENADE tutorial. `http://www-sop.inria.fr/tropics/tapenade/tutorial.html`, 2015. [Online; accessed 25-November-2015].

[105] Yang Wang, Siamak Akbarzadeh, and Jens-Dominik Müller. Stabilisation of discrete adjoint solvers for incompressible flow. In *22nd AIAA Computational Fluid Dynamics Conference*, page 2749, 2015.

[106] Shenren Xu, David Radford, Marcus Meyer, and Jens-Dominik Müller. Stabilisation of discrete steady adjoint solvers. *Journal of Computational Physics*, 299:175–195, 2015.

[107] M.S. Campobasso and M.B. Giles. Effect of flow instabilities on the linear analysis of turbomachinery aeroelasticity. *AIAA Journal of Propulsion and Power*, 19(4), 2003.

[108] JP Van Doormaal and GD Raithby. Enhancements of the simple method for predicting incompressible fluid flows. *Numerical heat transfer*, 7(2):147–163, 1984.

[109] M Darwish, F Moukalled. A unified formulation of the segregated class of algorithms for fluid flow at all speeds. *Numerical Heat Transfer: Part B: Fundamentals*, 37(1):103–139, 2000.

[110] R. I. Issa. Solution of the implicitly discretized fluid flow equations by operator splitting. *Journal of Computational Physics*, 62:40–65, 1986.

[111] Milovan Peric Joel H.Ferziger. *Computational Methods for Fluid Dynamics*. Springer, 1996.

[112] Howard C. Elman, David J. Silvester, and Andrew J. Wathen. Performance and analysis of saddle point preconditioners for the discrete steady-state navier-stokes equations. *Num. Math.*, 90(4):665–688, 2002.

[113] Michele Benzi, Michael Ng, Qiang Niu, and Zhen Wang. A relaxed dimensional factorization preconditioner for the incompressible navierstokes equations. *Journal of Computational Physics*, 230(16):6185–202, 2011.

[114] M Darwish, I Sraj, and F Moukalled. A coupled incompressible flow solver on structured grids. *Numerical Heat Transfer, Part B: Fundamentals*, 52(4):353–371, 2007.

[115] Marwan Darwish, Ihab Sraj, and Fadl Moukalled. A coupled finite volume solver for the solution of incompressible flows on unstructured grids. *Journal of Computational Physics*, 228(1):180–201, 2009.

[116] L Mangani, M Buchmayr, and M Darwish. Development of a novel fully coupled solver in openfoam: Steady-state incompressible turbulent flows. *Numerical Heat Transfer, Part B: Fundamentals*, 66(1):1–20, 2014.

[117] V. Przulj and B. Basara. A simplebased control volume method for compressible flows on arbitrary grids. In *Proceedings of the Thrity-Second Colloquium on the Law of Outer Space*, Wahington, DC, 2002. AIAA.

[118] J. Droniou, R. Eymard, T. Gallouët, and R. Herbin. A unified approach to mimetic finite difference, hybrid finite volume and mixed finite volume methods. *Mathematical Models and Methods in Applied Sciences*, 20(02):265–295, 2010.

[119] H. Sadok S. Duminil and D. Silvester. Fast solver for discretized navier-stokes problems using vector extrapolation. *Num. Algor.*, 66:89–104, 2014.

[120] Jun Zhang. Sparse approximate inverse and multilevel block ilu preconditioning techniques for general sparse matrices. *Applied Numerical Mathematics*, 35(1):67–86, 2000.

[121] Michele Benzi, Gene H Golub, and Jörg Liesen. Numerical solution of saddle point problems. *Acta numerica*, 14:1–137, 2005.

[122] David Ronald Kincaid and Elliott Ward Cheney. *Numerical analysis: mathematics of scientific computing*, volume 2. American Mathematical Soc., 2002.

[123] C.M. Klaij. On the stabilization of finite volume methods with co-located variables for incompressible flow. *Journal of Computational Physics*, 297:84 – 89, 2015.

[124] Philippe Traor, Yves Marcel Ahipo, and Christophe Louste. A robust and efficient finite volume scheme for the discretization of diffusive flux on extremely skewed meshes in complex geometries. *J. Comput. Phys.*, 228(14):5148 – 5159, 2009.

[125] S.-C. Xue and G.W. Barton. A finite volume formulation for transient convection and diffusion equations with unstructured distorted grids and its applications in fluid flow simulations with a collocated variable arrangement. *Computer Methods in Applied Mechanics and Engineering*, 253(0):146 – 159, 2013.

[126] G. H. Golub and C. F. Van Loan. *Matrix Computations*. JohnsHopkinsPress, Baltimore, MD, USA, second edition, 1989.

[127] *Hypre User's Manual*, 2012.

[128] P. Moinier, J.-D. Müller, and M. B. Giles. Edge-based multigrid schemes and preconditioning for hybrid grids. *AIAA Journal*, 40(10):1954–60, 2002.

[129] D. Mavriplis. Multigrid strategies for viscous flow solvers on unstructured meshes. Technical Report 98-6, ICASE, January 1998.

[130] U. Trottenberg, C.W. Oosterlee, and A. Schüller. *Multigrid*. Acdemic Press, 2001.

[131] K. Stüben. A review of algebraic multigrid. *Journal of Computational and Applied Mathematics*, 128:281309, 2001.

[132] A. Thom and C. J. Apelt. Note on the convergence of numerical solutions of the navier-stokes equations. Reports and memoranda no. 3061, Great Britain Aeronautical Research Council, 1958.

[133] R. I. Issa I. Demirdzic, A. D. Gosman and M. Peric. A calculation procedure for turbulent flow in complex geometries. *Computation Fluids*, 15:251273, 1987.

[134] M. Peric. Analysis of pressure velocity coupling on non-orthogonal grids. *Numerical Heat Transfer, Part B*, 17:63–82, 1990.

[135] R. H. Yen and C. H. Liu. Enhancement of the simple algorithm by an additional explicit correction step. *Numerical Heat Transfer, Part B*, 17:127141, 1993.

[136] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[137] Jamshid A Samareh. A survey of shape parameterization techniques. 1999.

[138] J.A. Samareh. Survey of shape parameterization techniques for high-fidelity multidisciplinary shape optimization. *AIAA Journal*, 39(5):877–884, 2001.

[139] J.A. Samareh. Aerodynamic shape optimization based on free-form deformation. *AIAA*, 4630:2004, 2004.

[140] Raphael T Haftka and Ramana V Grandhi. Structural shape optimizationa survey. *Computer methods in applied mechanics and engineering*, 57(1):91–106, 1986.

[141] Yunliang Ding. Shape optimization of structures: a literature survey. *Computers & Structures*, 24(6):985–1004, 1986.

[142] Majid Hojjat, Electra Stavropoulou, and Kai-Uwe Bletzinger. The vertex morphing method for node-based shape optimization. *Computer Methods in Applied Mechanics and Engineering*, 268:494 – 513, 2014.

[143] A. Jameson and A. Vassberg. Studies of alternative numerical optimization methods applied to the brachistochrone problem. *Computational Fluid Dynamics Journal*, 9(3), 2000.

[144] Stephan Schmidt, Caslav Ilic, Nicolas Gauger, and Volker Schulz. Shape gradients and their smoothness for practical aerodynamic design optimization. DFG SFB 1253 Preprint-Number SPP1253-10-03 http://, Universität Erlangen, April 2008.

[145] A. Jaworski and J.-D. Müller. Toward modular multigrid design optimisation. *Lecture Notes in Comp. Sci. Eng.*, 64:281–291, 2008.

[146] Karsten Eppler, Stephan Schmidt, Volker Schulz, and Caslav Ilic. Preconditioning the pressure tracking in fluid dynamics by shape hessian information. *Journal of Optimization Theory and Applications*, 141(3):513–531, 2009.

[147] Arthur Stück and Thomas Rung. Adjoint rans with filtered shape derivatives for hydrodynamic optimisation. *Computers & Fluids*, 47(1):22–32, 2011.

[148] G. Yu, J.-D. Müller, and D. Jones. CAD-based shape optimisation using adjoint sensitivities. *Computers and Fluids*, 46(1):512–16, 2011.

[149] E. Hardee, K.H. Chang, J. Tu, K.K. Choi, I. Grindeanu, and X. Yu. A cad-based design parameterization for shape optimization of elastic solids. *Advances in Engineering Software*, 30:185–199, 1999.

[150] T.T. Robinson, C.G. Armstrong, H.S. Chua, C. Othmer, and T. Grahs. Optimizing parameterized CAD geometries using sensitivities based on adjoint functions. *Computer-Aided Design & Applications*, 9(3):253–268, 2012.

[151] L. Piegl and W. Tiller. *The NURBS Book*. 2nd edition. Springer, 1997.

[152] Wolfram Jahn. Manual for the computation of surface derivatives for shape optimisation using boundary representation, 2011.

[153] Michael J Pratt. Introduction to iso 10303the step standard for product data exchange. *Journal of Computing and Information Science in Engineering*, 1(1):102–103, 2001.

[154] Richard P Dwight. Robust mesh deformation using the linear elasticity equations. In *Computational Fluid Dynamics 2006*, pages 401–406. Springer, 2009.

[155] Eric J Nielsen and W Kyle Anderson. Recent improvements in aerodynamic design optimization on unstructured meshes. *AIAA journal*, 40(6):1155–1163, 2002.

[156] Zhi Yang and Dimitri J Mavriplis. Mesh deformation strategy optimized by the adjoint method on unstructured meshes. *AIAA journal*, 45(12):2885–2896, 2007.

[157] Hrvoje Jasak and HG Weller. Application of the finite volume method and unstructured meshes to linear elasticity. *International journal for numerical methods in engineering*, 48(2):267–287, 2000.

[158] Robert T Biedron and James L Thomas. Recent enhancements to the fun3d flow solver for moving-mesh applications. *AIAA Paper*, 1360:2009, 2009.

[159] E Fares and W Schröder. A differential equation for approximate wall distance. *International journal for numerical methods in fluids*, 39(8):743–762, 2002.

[160] Garbis Hovannes Keulegan and Karl Hilding Beij. *Pressure losses for fluid flow in curved pipes*. US Government Printing Office, 1937.

[161] Philip AJ Mees, K Nandakumar, and JH Masliyah. Instability and transitions of flow in a curved square duct: the development of two pairs of dean vortices. *Journal of Fluid Mechanics*, 314:227–246, 1996.

[162] S.A. Berger, L. Talbot, and L.S.Yao. Flow in curved pipes. *Annual Review of Fluid Mechanics*, 15(1):461–512, 1983.

[163] WR Dean and JM Hurst. Note on the motion of fluid in a curved pipe. *Mathematika*, 6(01):77–85, 1959.

[164] Jon Johnson and Markus Busch. Engineering aspects of reverse osmosis module design. *Desalination and Water Treatment*, 15(1-3):236–248, 2010.

[165] J Schwinge, DE Wiley, and DF Fletcher. Simulation of the flow around spacer filaments between narrow channel walls. 1. hydrodynamics. *Industrial & engineering chemistry research*, 41(12):2977–2987, 2002.

[166] J Schwinge, DE Wiley, and DF Fletcher. Simulation of unsteady flow and vortex shedding for narrow spacer-filled channels. *Industrial & engineering chemistry research*, 42(20):4962–4977, 2003.

[167] EA Spiegel and G Veronis. On the boussinesq approximation for a compressible fluid. *The Astrophysical Journal*, 131:442, 1960.

[168] Vtor Geraldes, Viriato Semio, and Maria Norberta de Pinho. Flow and mass transfer modelling of nanofiltration. *Journal of Membrane Science*, 191(12):109 – 128, 2001.

[169] M.F. Gruber, C.J. Johnson, C.Y. Tang, M.H. Jensen, L. Yde, and C. Hlix-Nielsen. Computational fluid dynamics simulations of flow and concentration polarization in forward osmosis membrane systems. *Journal of Membrane Science*, 379(12):488 – 495, 2011.

[170] Warren E. Stewart R. Byron Bird and Edwin N. Lightfoot. *Transport Phenomeno*. John Wiley & Sons, 2 edition, 2002.

[171] Abraham S Berman. Laminar flow in channels with porous walls. *Journal of Applied physics*, 24(9):1232–1235, 1953.

[172] Sandeep K. Karode. Laminar flow in channels with porous walls, revisited. *Journal of Membrane Science*, 191(12):237 – 241, 2001.

[173] D.F. Fletcher and D.E. Wiley. A computational fluids dynamics study of buoyancy effects in reverse osmosis. *Journal of Membrane Science*, 245(12):175 – 181, 2004.

[174] A. Alexiadis, D.E. Wiley, A Vishnoi, R.H.K. Lee, D.F. Fletcher, and J. Bao. {CFD} modelling of reverse osmosis membrane flow and validation with experimental results. *Desalination*, 217(13):242 – 250, 2007.

[175] Dianne E Wiley and David F Fletcher. Techniques for computational fluid dynamics modelling of flow in membrane channels. *Journal of Membrane Science*, 211(1):127–137, 2003.

[176] Frank Lipnizki and Robert W Field. Mass transfer performance for hollow fibre modules with shell-side axial feed flow: using an engineering approach to develop a framework. *Journal of Membrane Science*, 193(2):195–208, 2001.